

Tutorial: Configuring Payloads for XR Vessels

By Ryan Kingsbury "Countdown84"

Introduction

This tutorial provides a detailed explanation of the payload management system used by Doug Beachy's XR-series vessels (namely, the XR5 Vanguard and XR2 Ravenstar). It begins with a conceptual discussion of how these vessels handle payloads and perform collision detection, followed by two step-by-step examples of how to configure Orbiter vessels as payloads. It concludes with some special notes for configuring Spacecraft3.dll vessels and vessels with custom .dll modules as payloads.

Requirements

To follow this tutorial, you will need Orbiter 060929-P1 Edition (of course), as well as the XR5 Vanguard available at Doug Beachy's Orbiter Page:

<http://www.dougsorbiterpage.com/index-3.html>

Note that the XR5 requires UMMU 1.5 and OrbiterSound 3.5, available at DanSteph's page:

<http://orbiter.dansteph.com/index.php?disp=d>

Another useful (but not required) utility is the Mesh Wizard by José Pablo Luna Sánchez, available at Orbithangar:

<http://orbithangar.com/searchid.php?ID=2740>

In order to make thumbnail images of your payload, you will need the screen capture module for Orbiter, available at Orbithangar:

<http://www.orbithangar.com/searchid.php?ID=3279>

You will also need an image-manipulation program capable of cropping, resizing, and saving .jpg and .bmp images. I highly recommend Irfanview, but the choice is yours:

<http://www.irfanview.us/>

This tutorial assumes you have a fairly good knowledge of how to use the XR5 to select, spawn, grapple, and deploy payloads. Also, a working knowledge of vectors and vector notation is very important. If you are uncomfortable with vectors and vector notation I highly recommend the "Vectors for Dummies" tutorial at

<http://orbithangar.com/searchid.php?ID=3178>.

Contents

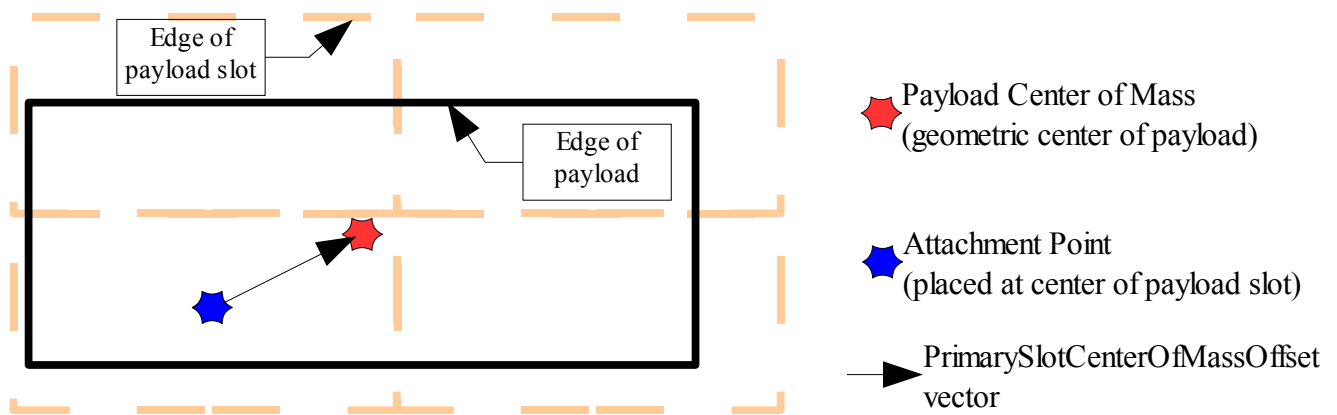
Introduction.....	1
Requirements.....	1
Chapter 1: Payload Management Logic.....	3
Chapter 2: A Basic Example – the Carina Satellite.....	3
Step 1: Get to know the Carina.....	4
Step 2: Modify the Configuration File.....	4
Step 3: Test.....	6
Step 4: Adjust Attachment Parameters.....	7
Step 5: Check for Bounce.....	9
Chapter 3: An Oddly-Shaped Payload – the Hubble Telescope.....	10
Step 1: Get to know the Hubble.....	10
Step 2: Modify the Configuration File.....	11
Step 3: Determine the Mesh Size.....	11
Step 4: Bypass Collision Detection (Cheat).....	13
Step 5: Adjust the Attachment Parameters.....	14
Step 7: Tweak the Collision Detection for Other Payloads.....	17
Step 8: Make A Thumbnail Image.....	18
Step 9: Check for Bounce.....	20
Step 10: Final Test.....	20
Chapter 4: Notes for Vessels with .dll Modules (including Spacecraft3.dll).....	21
Additional Considerations for Payloads with .dll Modules.....	21
Special Issues with Spacecraft3.dll.....	21
Acknowledgements.....	21
Disclaimer.....	21
Comments / Questions.....	21
Appendix: Quick Reference Guide for Configuration File Parameters.....	22

Chapter 1: Payload Management Logic

The XR payload management system divides a vessel's cargo bay into a discrete number of “slots,” or rectangular spaces. Each slot has an attachment point located in the center of it. The payload manager examines the configuration file for each payload to determine its dimensions (length, width, height), and then treats that payload as a box. *The payload manager does not examine the mesh of the payload – all vessels are handled as “boxes” regardless of shape.*

Every payload's configuration file defines one unique attachment point, which can be located anywhere within the payload's “box.” When a payload is grappled into a slot, the payload's attachment point is placed exactly in the middle of the slot.

But, the payload's attachment point does not have to be located in the middle of the payload. In fact, the payload management system allows you to specify an offset between the center of mass of the payload and the attachment point. The center of mass (for our purposes) is *always* located at the geometric center of the “box” defined by the payload's dimensions. Similarly, the payload's attachment point is *always* located at the geometric center of the payload slot it is grappled into. A schematic of this system is shown below.



These two facts allow the payload manager to compute how many slots a payload will occupy. For example, suppose that each slot is 10 m long, and you have a payload that is 15 m long. Say the payload's attachment point is located 5 m from the front of the payload, while of course the center of mass is located 7.5 m from the front. If the payload is grappled, the attachment point will be placed in the middle of a 10 m slot. As such, the front edge of the payload will just fit inside the slot ($10/2 = 5$ m, which is the distance from the attachment point to the front), while the back end will extend 5 m beyond the slot (15 m - 10 m = 5 m). Your payload is then said to occupy *two* payload slots.

The XR payload management system performs these calculations in all three dimensions, so that it can determine how many slots are occupied by any payload. In the two-dimensional figure above, the payload would occupy a total of four slots (2 horizontal, 2 vertical). Note, however, that it is only grappled into *one* slot (the one containing the attachment point). This is referred to as the “primary” payload slot.

All of this is probably best illustrated with an example.

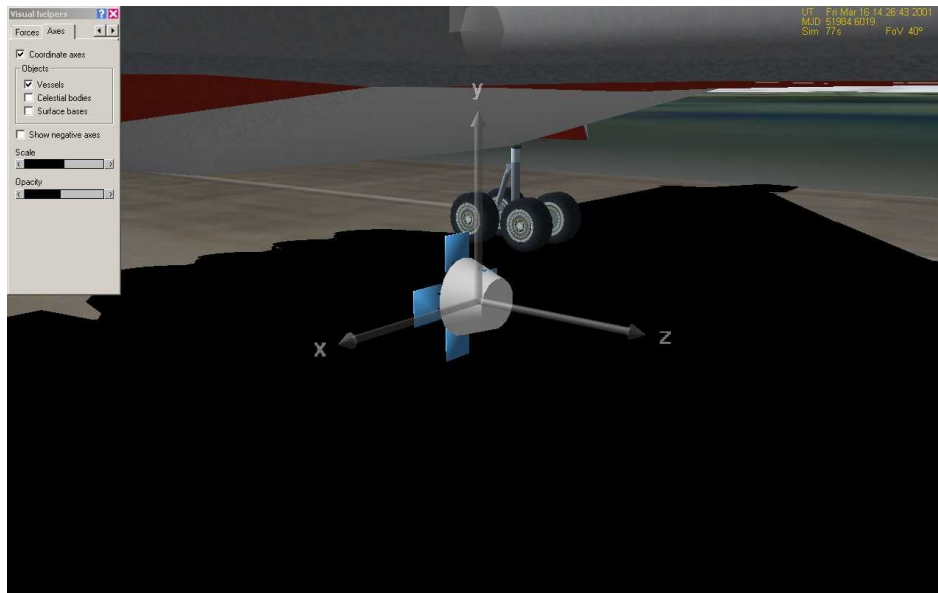
Chapter 2: A Basic Example – the Carina Satellite

This section describes a bare-minimum payload configuration for the Carina satellite included in the stock Orbiter distribution. There are many additional optional parameters that can be specified in the configuration file, but here we will only cover those necessary for a working payload.

Step 1: Get to know the Carina

Use Orbiter scenario editor (CTRL+F4) to create a scenario with a Carina satellite on the runway next to the XR5. Your purpose here is to get a feel for the relative size of the payload and start to think about how many slots it might occupy, how you will fit it into the bay, etc.

You should also take note of how the payload's coordinate system is oriented. Press CTRL+F9 to bring up the Visual Helpers dialog. Select the “Axes” tab at the top and check the box “Coordinate Axes.” The axes of all vessels are now displayed. As you can see, the standard orientation for an Orbiter vessel is for the +z axis to point forward (along the nose), while the +y points up and the +x points sideways. *These coordinate directions are properties of the mesh.* So, now matter which way you're flying or moving, the +z axis will *always* point out the round top of the Carina, and the +y will point along one of the solar panels.



It's important to note how the axes are oriented relative to the mesh, because everything that we will specify in the payload configuration file involves vectors that relate either to the payload's coordinate system or to the XR5's. If you don't know which way +x, +y, and +z are, you will get VERY frustrated trying to position things correctly!

Step 2: Modify the Configuration File

Find the Carina's .cfg file. Navigate to your Orbiter/Config/Vessels/ folder and open up “Carina.cfg” with notepad. We now need to add several things to this file in order to make the XR5 recognize it as a payload.

Add an attachment point

You will need to define a new attachment point in the section between “BEGIN_ATTACHMENT” and “END_ATTACHMENT.” Our new attachment point has to be formatted like so:

“P (x,y,z) (ROT) (DIR) XRCARGO”

Where (x,y,z), (ROT), and (DIR) are all vectors.

(x,y,z) defines *where* the payload attachment point will be located. This is a vector in the payload vessel's coordinate system. Setting it to “0 0 0” means that the attachment point will be located at the vessel's center of mass. This is fine for now.

(ROT) defines *which way is up* when the vessel is attached. Normally, the +y axis is up for a vessel, but this vector allows the vessel to be rotated another way when attached. Leave it at “0 1 0” (indicating +y axis) for now.

(DIR) defines *which way points forward* when the vessel is attached. Normally, the +z axis is forward, but you can point the payload another way with this vector. Leave it at “0 0 1” (indicating +z axis) for now.

XRCARGO is an identifier for the attachment point that specifies it as the one to be used when grappling into an XR vessel.

Enable XRPAYLOAD

Add the line “XRPayloadEnabled=TRUE” to the bottom of the config file. This is what the XR vessels look for to tell them that your vessel is an XR payload. **MAKE SURE TO INCLUDE THIS LINE!**

Point to the Correct Attachment Point

Add the line “AttachmentPointIndex=2” to the end of the config file. This line points tells the XR payload management system which of the attachment points to use for grappling into XR vessels. *Note that the index starts counting at 0, not 1.* So if your “XRCARGO” point is the third attachment point in the list, this value should be set to “2” (0,1,2)

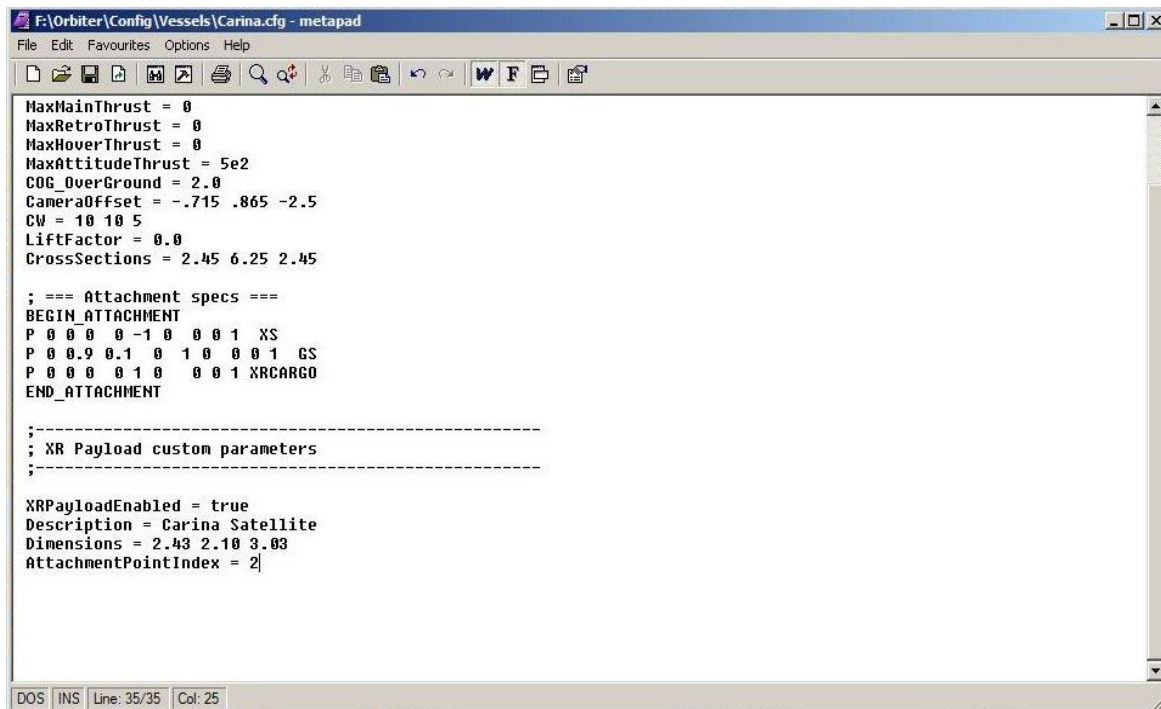
Add a Description

When your payload appears in the XR5 (or XR2) payload manager window, it will have a text description. Specify this by adding the line “Description = Carina Satellite” to the file.

Specify Dimensions

In order for the payload manager to perform collision-detection calculations as described above, it has to know how big of a “box” to place around the payload. These are given as a vector with the line “Dimensions = X Y Z” where X,Y, and Z are the width, height, and length of your payload in meters. There are several ways to determine what the actual dimensions are. In this tutorial, we will 1) guess that it is small to make sure we can grapple it easily and 2)compare it with the standard XR5 cargo containers to estimate the size. So, for now, set X, Y, and Z to 2.0 meters each, because these are less than the dimensions of an XR5 payload slot. In the screenshot below I have arbitrarily used 2.43, 2.10, and 3.03 m because these were leftover from another payload :-). As long as it is less than one slot it doesn't matter.

When all is said and done, your configuration file should look like this:



```
F:\Orbiter\Config\Vessels\Carina.cfg - metapad
File Edit Favourites Options Help

MaxMainThrust = 0
MaxRetroThrust = 0
MaxHoverThrust = 0
MaxAttitudeThrust = 5e2
COG_OverGround = 2.0
CameraOffset = -.715 .865 -2.5
CW = 10 10 5
LiftFactor = 0.0
CrossSections = 2.45 6.25 2.45

; === Attachment specs ===
BEGIN_ATTACHMENT
P 0 0 0 0 -1 0 0 0 1 XS
P 0 0.9 0.1 0 1 0 0 0 1 GS
P 0 0 0 0 1 0 0 0 1 XRCARGO
END_ATTACHMENT

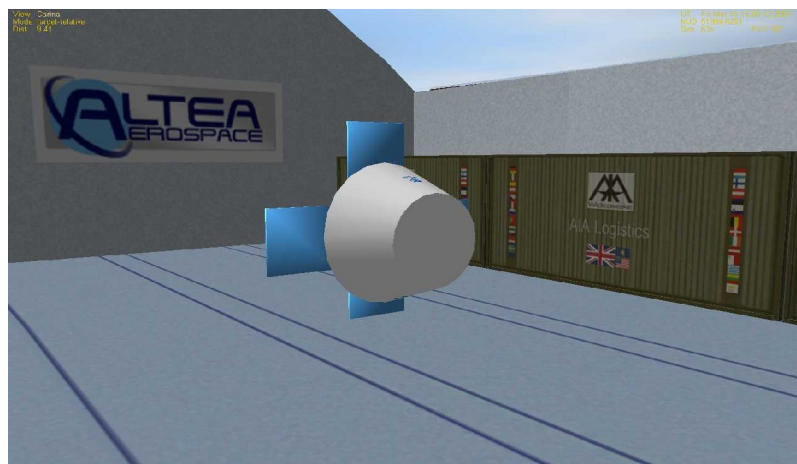
; -----
; XR Payload custom parameters
; -----

XRPayloadEnabled = true
Description = Carina Satellite
Dimensions = 2.43 2.10 3.03
AttachmentPointIndex = 2

DOS INS Line: 35/35 Col: 25
```

Step 3: Test

Save your configuration file (make sure to close the Orbiter launchpad or else it will not allow you to save the file). Now re-open Orbiter launchpad and start your favorite XR5 scenario. Bring up the payload manager by pressing ALT+B. If you scroll through the list of available payloads you should see one called “Carina.” Go ahead and spawn one of these into a payload slot of your choice. For size comparison, you might spawn a bunch of “AIA_Logistics” modules into the bay next to it. You should see something like this:



Congratulations! You have a working XR5 payload. At this point you should take note of a couple of things. First, the satellite is sideways – just like it was on the ground. This is because we defined the DIR and ROT vectors of the attachment point to correspond with a “normal” vessel orientation - +y is up and +z is forward. Second, the satellite looks like it's roughly the same width and height as a standard payload slot (about 2.5 m), but much shorter in length.

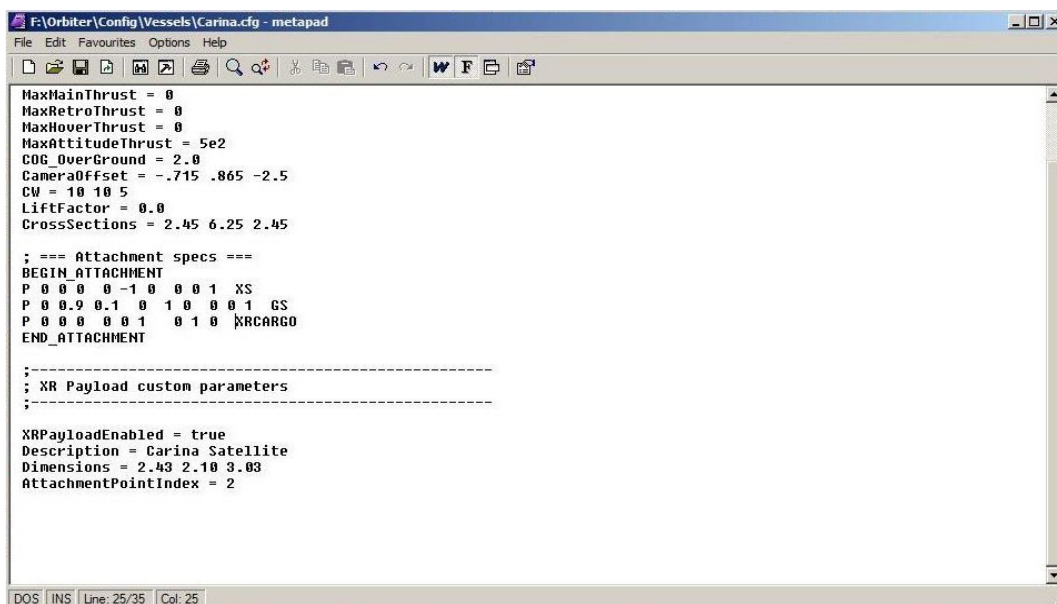
Step 4: Adjust Attachment Parameters

Now it's time to make adjustments to the configuration parameters so that the Carina “behaves” better in the bay.

Payload Orientation

First, we need to fix the satellite so it doesn't sit on its side (that solar panel is fragile after all!). The way to approach this is to figure out which directions *in the payload's coordinate system* we want to point forward and up. We know that the front round part of the Carina is facing the +z direction, but we'd like that to be the top. We also know that the four solar panels point in the x and y directions, and since the satellite is symmetrical it doesn't really matter which of those points forward.

Go back to the config file and locate the XRCARGO attachment point line we added earlier. To change the orientation of the Carina in the bay, we will modify the ROT and DIR vectors so that +z is up and +y is forward. This means that ROT = 0 0 1 and DIR = 0 1 0. Save the file with the changes and launch Orbiter again to test it. Make sure you've closed Orbiter Launchpad to avoid an error when saving. The config file and result should look like:



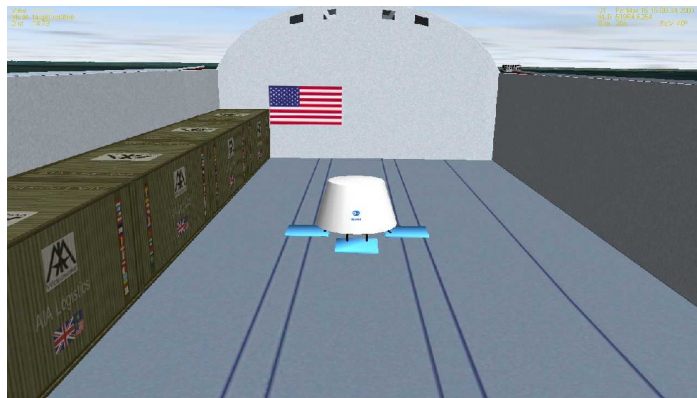
```
F:\Orbiter\Config\Vessels\Carina.cfg - metapad
File Edit Favourites Options Help
MaxMainThrust = 0
MaxRetroThrust = 0
MaxHoverThrust = 0
MaxAttitudeThrust = 5e2
COG_OverGround = 2.0
CameraOffset = -.715 .865 -2.5
CM = 10 10 5
LiftFactor = 0.0
CrossSections = 2.45 6.25 2.45

; === Attachment specs ===
BEGIN_ATTACHMENT
P 0 0 0 0 -1 0 0 0 1 XS
P 0 0 0 0 1 0 0 0 1 GS
P 0 0 0 0 0 1 0 1 0 XRCARGO
END_ATTACHMENT

; -----
; XR Payload custom parameters
; -----

XRPayloadEnabled = true
Description = Carina Satellite
Dimensions = 2.43 2.10 3.03
AttachmentPointIndex = 2

DOS INS Line: 25/35 Col: 25
```



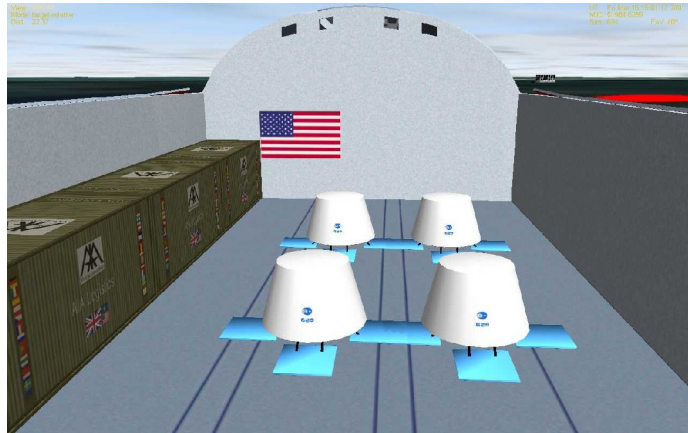
[Aside] If you have tried editing your own payloads before you may have noticed the following lines in the example configuration file included with the XR5:

“WARNING: attachment *direction* must be (0 1 0) and attachment *rotation* must be (0 0 1)! The bay collision detection code expects this.”

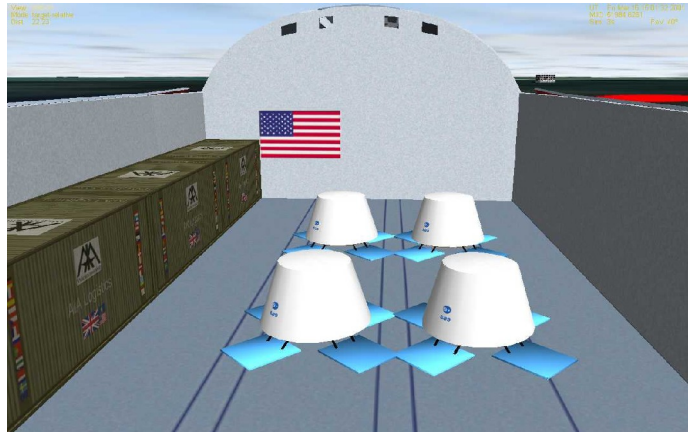
In fact, we just changed the direction and rotation vectors to other values without a problem. The reason this warning is included is that the payload dimensions have to be specified **in the XR5's coordinate system**, not the payload's. For example, imagine a payload that is 1m wide (x), 1m tall (y), and 10 m long (z). If we put that into the XR5 and rotate it, it might seem to be 10m wide, 1m tall, and 1m long. The payload management code does not automatically transform the dimensions provided in the configuration file, it assumes that they are specified relative to the XR5's coordinate system. X is width, Y is height, Z is length *relative to the XR5*.

So in sum, it's perfectly fine to change the DIR and ROT vectors to values other than 0 1 0 and 0 0 1, as long as you are careful to modify the DIMENSIONS parameter accordingly.

But now there's another problem. It looks like the solar panels are a little too wide for a standard slot. If you spawn a couple of Carinas next to one another you'll see that they actually run into one another:



One solution is to rotate the satellite so that it sits diagonally in its slot. To do this, we go back to the configuration file and modify the DIR vector (which way is forward?) so that a direction halfway between +x and +y is pointing forward. This requires that $DIR = 0.707\ 0.707\ 0$. And now the satellites fit very nicely together:



Payload Size

It's important to get the dimensions right so that the collision detection calculations will work correctly and we won't have Carinas running into other payloads. For our first attempt we arbitrarily assigned dimensions to the Carina. Now that we have it in the bay, we can see that the satellite does in fact fit inside one payload slot, so any dimensions that are less than one slot will be appropriate.

It's very important to be mindful of a subtlety here. The “Dimensions =” line specified in the config file specifies dimensions **in the XR5's coordinate system** (see the aside above). As shown in the screenshot above, it looks like the width of the Carina, from corner-to-corner of the solar panels, is almost exactly that of a “standard” payload slot – 2.43 m. It's a square, so the length would be the same. For the height, it looks approximately 2/3 the height of an “AIA_Logistics” module, which is 2.6m. So, by estimating the size relative to a standard payload container, I chose to go back and modify the arbitrary dimensions in the config file, using 2.2 m for width and length, and 2.0 m for height.

Step 5: Check for Bounce

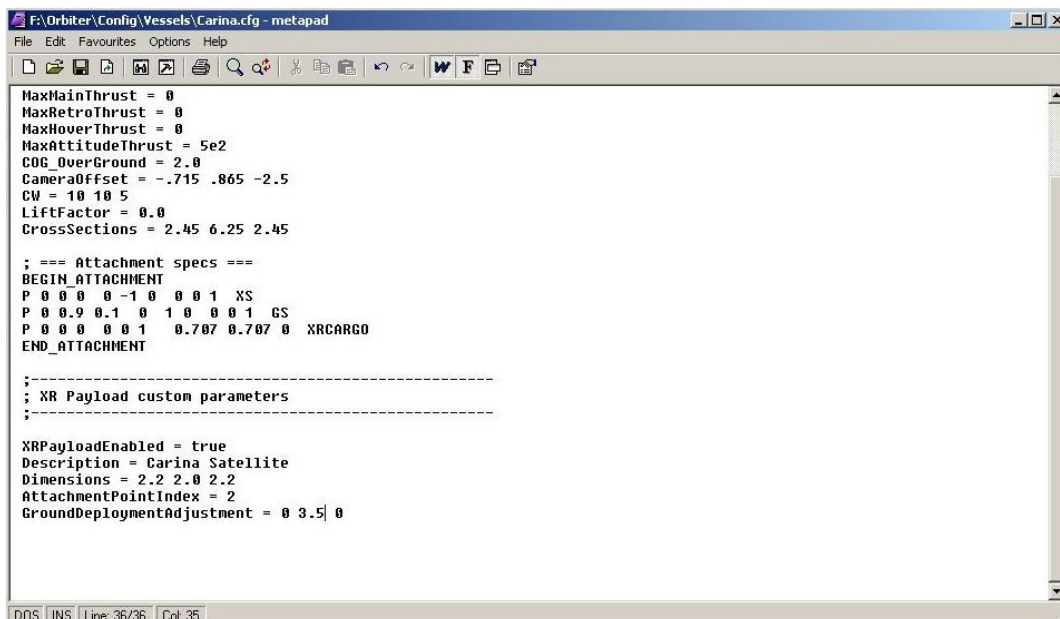
The final step in this process is to check for the “bounce bug” on payload deployment. The bounce bug occurs when a vessel finds itself below the ground (or, when its touchdown points are below the ground). If this occurs, the vessel bounces into the air, sometimes a few meters, sometimes thousands. This can occasionally occur when you deploy a payload onto the ground from the XR5. However, there is a way to adjust for.

To test, launch an Orbiter scenario in which you have the Carina grappled in the payload bay. Go to an external view (so that you can see the area where payloads deploy) and press CTRL+ALT+U to deploy the Carina. You will see that it bounces a little bit.

To correct this, add the line “GroundDeploymentAdjustment = x y z” to the configuration file. As before, x y z is a vector defining an offset relative to the location the payload is normally deployed to. For the bounce bug, the only value you ever need to change is y. For example, 0 5 0 will deploy the payload 5 m higher than normal, hopefully avoiding the bounce bug.

Experiment with different values until you find one that deploys the payload close to the ground, but does not bounce. If you set it too high the payload will deploy up in the air and fall. I found 3.5 m to be about perfect for the Carina.

The finalized Carina.cfg file now looks like this:



```

F:\Orbiter\Config\Vessels\Carina.cfg - metapad
File Edit Favourites Options Help
MaxMainThrust = 0
MaxRetroThrust = 0
MaxHoverThrust = 0
MaxAttitudeThrust = 5e2
COG_OverGround = 2.0
CameraOffset = -.715 .865 -2.5
CW = 10 10 5
LiftFactor = 0.0
CrossSections = 2.45 6.25 2.45

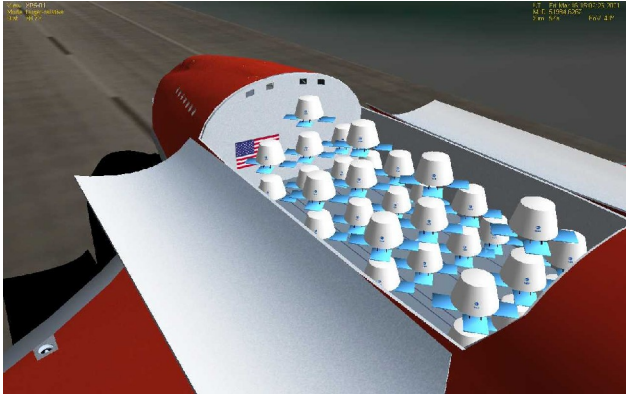
; === Attachment specs ===
BEGIN ATTACHMENT
P 0 0 0 0 -1 0 0 0 1 XS
P 0 0.9 0.1 0 1 0 0 0 1 GS
P 0 0 0 0 0 1 0.707 0.707 0 XRCARGO
END_ATTACHMENT

; -----
; XR Payload custom parameters
; -----

XRPayloadEnabled = true
Description = Carina Satellite
Dimensions = 2.2 2.0 2.2
AttachmentPointIndex = 2
GroundDeploymentAdjustment = 0 3.5 0
  
```

Congratulations! You now have a fully XR-compatible Carina!

Note that even though we used the XR5 to do all the configuration, Carina works equally well with the XR2!

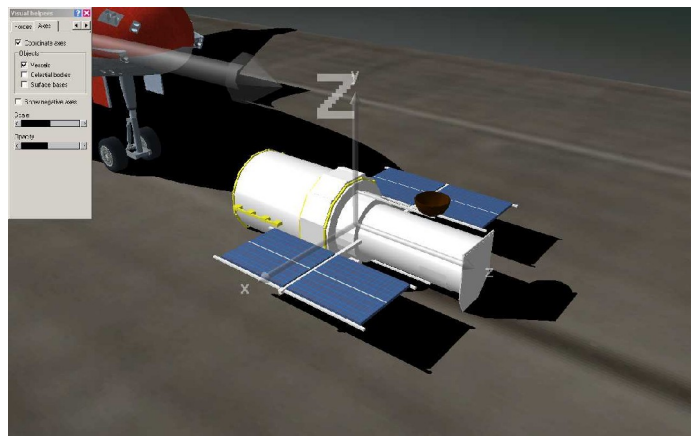


Chapter 3: An Oddly-Shaped Payload – the Hubble Telescope

The Carina is fairly straightforward, as its shape is somewhat square and it easily fits inside one payload slot. In this tutorial we'll look at a much larger, more oddly-shaped payload – the Hubble Space Telescope included in the stock Orbiter distribution. Along the way, we will cover a few more advanced details about payload configuration.

Step 1: Get to know the Hubble

We'll follow very much the same procedure as with the Carina. Open up an XR5 scenario with Orbiter, and use the built-in scenario editor to place the Hubble next to it (it's called “hst”). The, turn on the coordinate axes so you can see how the mesh is oriented.



You may notice that the top (+y) side of the telescope is a little plain. In fact, most of the details (NASA logo, attachment points, etc.) are located on the bottom of the mesh.

Step 2: Modify the Configuration File

The next step is to set up the basic parameters of the configuration file to make the Hubble work as a payload. Open up “Orbiter/Config/Vessels/hst.cfg” with notepad. As described above, you'll need to add the following parameters to the config file:

1. an attachment point
2. XRPayloadEnabled = TRUE
3. AttachmentPointIndex
4. Description
5. Dimensions

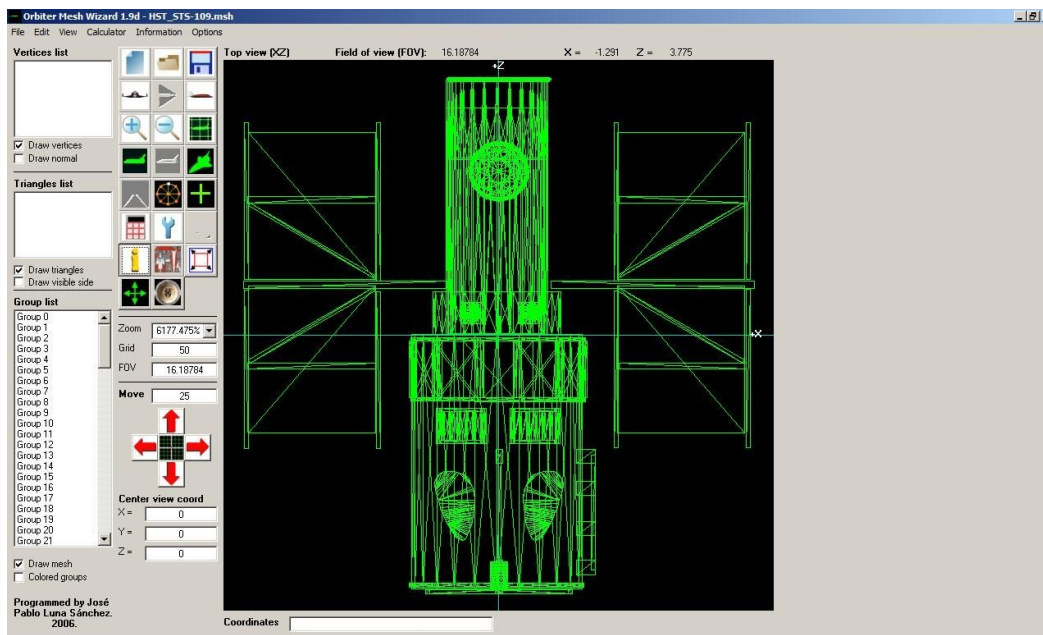
To get things working, place the attachment point coordinates at the Hubble's center of mass, with the +z axis forward and the +y axis pointing upward:

```
P 0 0 0 0 1 0 0 0 1 XRCARGO
```

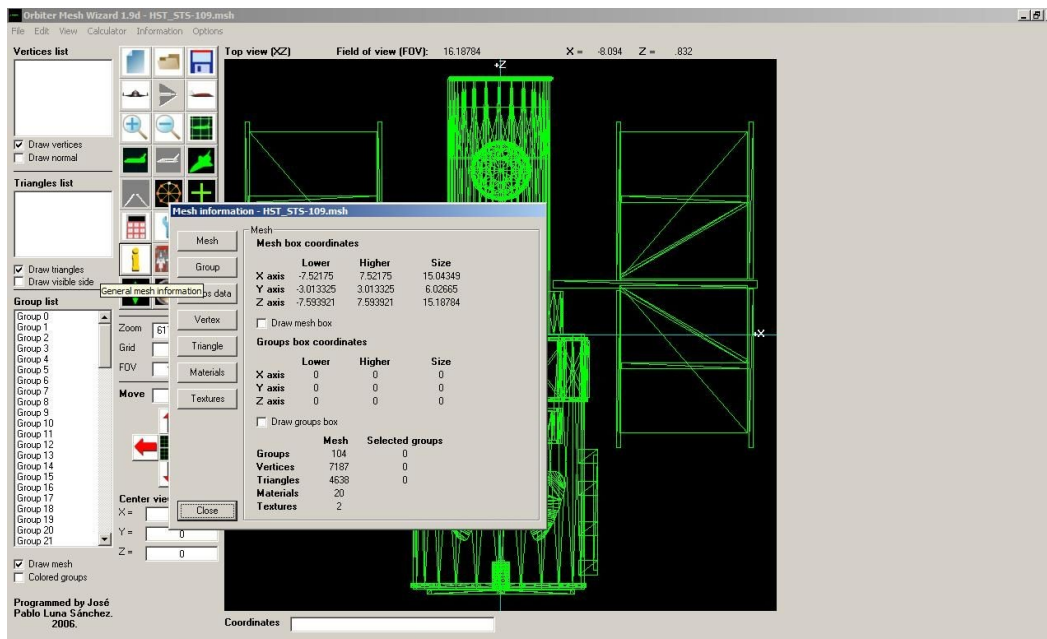
Since this is the fourth attachment point, the index will be 3 (0,1,2,3). Add any description you like, and proceed to the next step to figure out the dimensions.

Step 3: Determine the Mesh Size

With a small payload like the Carina, it's easy enough to estimate the size by comparing with a payload container. For something large like this, though, we'll use MeshWizard to actually measure the size of the mesh. Start MeshWizard, then open the file “Orbiter/Meshes/HST-109-d.msh.” You'll see the wireframe of the mesh in the window:



Click the big “i” to pull up a mesh information screen. This will show you the exact dimensions of the Hubble.



Toward the top of the information window you'll see three columns. The rightmost ("size") gives you the width (x), height (y), and length (z) of the mesh in meters. You can add these values directly into the configuration file with the "Dimensions = x y z" line.

With all these values added, the new configuration file should look like this:

```

* F:\Orbiter\Config\Vessels\hst.cfg - metapad
File Edit Favourites Options Help

; === Configuration file for HST Hubble Space Telescope ===
ClassName = HST
Module = HST
MeshName = HST_STS-109
;MeshName = HST
ImageBmp = Images\Vessels\Default\HST.bmp

; === Attachment specs ===
BEGIN_ATTACHMENT
P 0.1 2.7 0 0 1 0 0 0 1 XS
P 0.85 -2.75 0.67 0 -1 0 0 0 -1 GS
P -0.85 -2.75 0.67 0 -1 0 0 0 -1 GS
P 0 0 0 0 0 1 0 1 0 XRCARGO
END_ATTACHMENT

;-----
; XR Payload custom parameters
;-----

XRPayloadEnabled = true

Description = Hubble Space Telescope

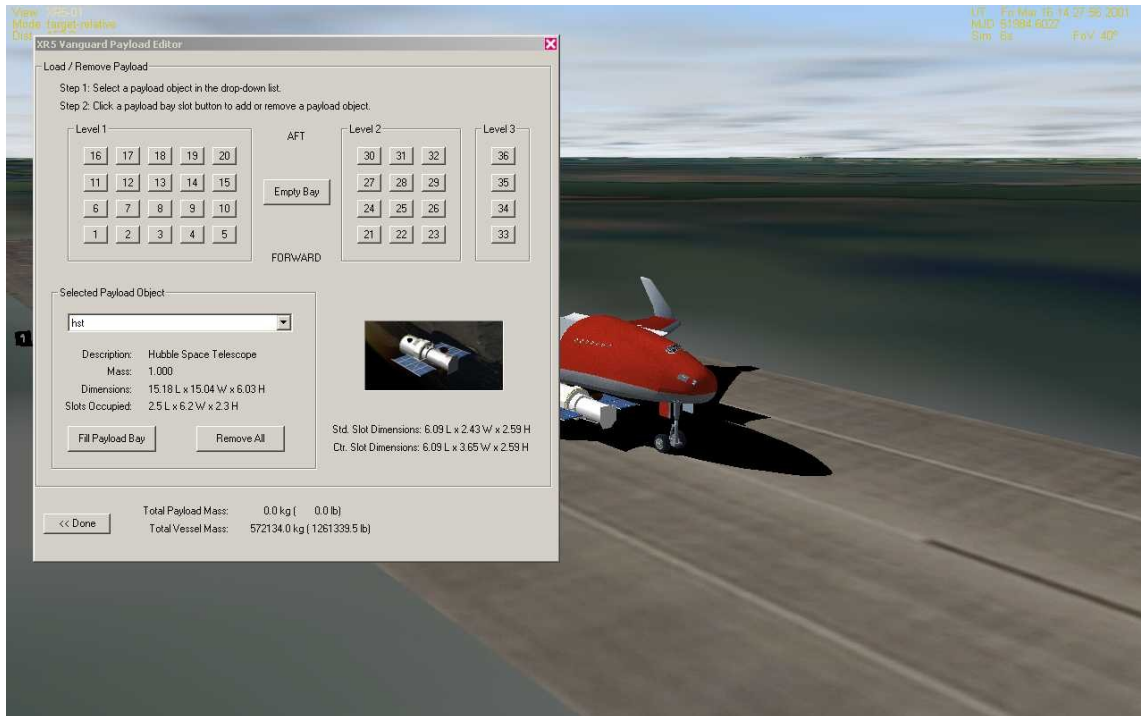
Dimensions = 15.04 6.03 15.18

AttachmentPointIndex = 3

```

Step 4: Bypass Collision Detection (Cheat)

Launch Orbiter again, into the scenario from Step 1 where the Hubble is sitting on the tarmac next to the XR5. Open the XR5 payload manager by pressing ALT+B. In the drop-down list, you should find an option called “hst.” Select it, and your description and configuration parameters will appear in the window (you won't see the screenshot; we haven't defined that yet :-)



If you look closely at the “Slots Occupied” field, you will notice that with the dimensions we entered in the last step, the Hubble will occupy 2.5 slots in length, 6.2 slots in width, and 2.3 slots in height. Unfortunately, the XR5 bay is only 5 slots wide, so there's no way the Hubble can fit.

Fortunately, the payload management system let's us “cheat” if we want to force something to fit. We need to explicitly define which XR5 slots the Hubble can be grappled into. Note that this *overrides* the collision-detection code for the purposes of determining whether or not the vessel can be grappled. In other words, if you tell the XR5 that this payload will fit in slot 8, it doesn't second-guess you.

Go back to the configuration file and add two lines:

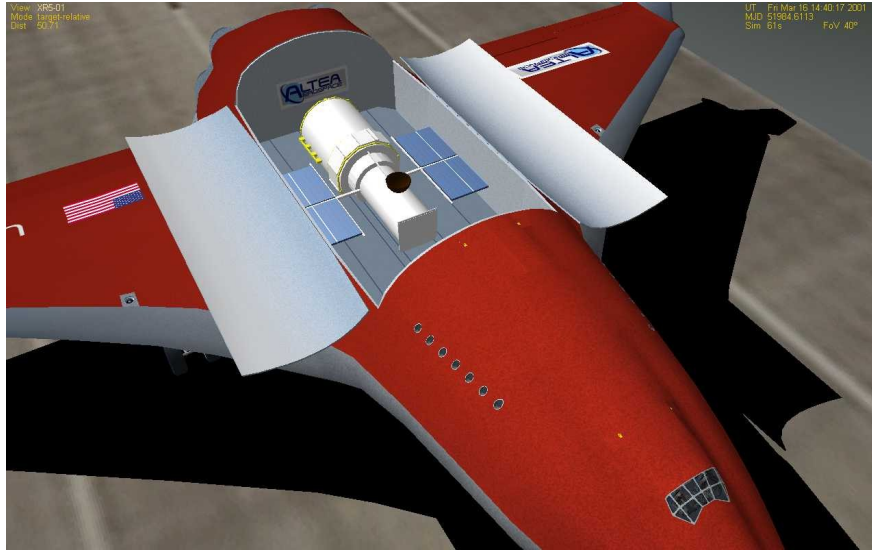
```
VesselsWithExplicitAttachmentSlotsDefined = XR5Vanguard
```

```
XR5Vanguard_ExplicitAttachmentSlots = 13
```

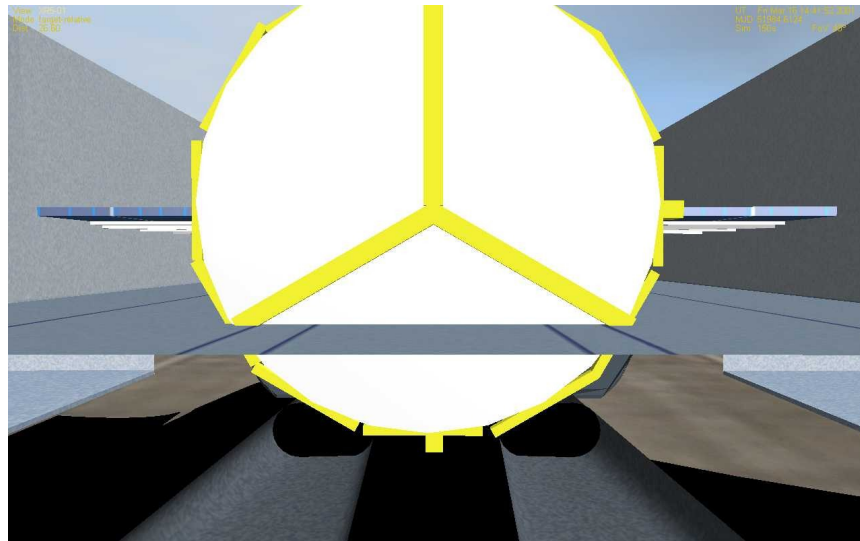
The first line tells the payload manager that we want to explicitly define which payload slots are “legal” for the Hubble to be grappled into. The second line specifies the exact slots. How do we know which to use? Take another look at the payload manager window in the screenshot above. You'll notice that all the payload bay slots are numbered. Since the Hubble is a big payload, I'm going to choose a center slot on the lower level. Note that you can specify multiple slots here (e.g. 13, 16).

Step 5: Adjust Attachment Parameters

Launch Orbiter again. This time, use the payload manager to try and grapple the Hubble into slot 13. If you've done things correctly, you should see the telescope in your payload bay like so:

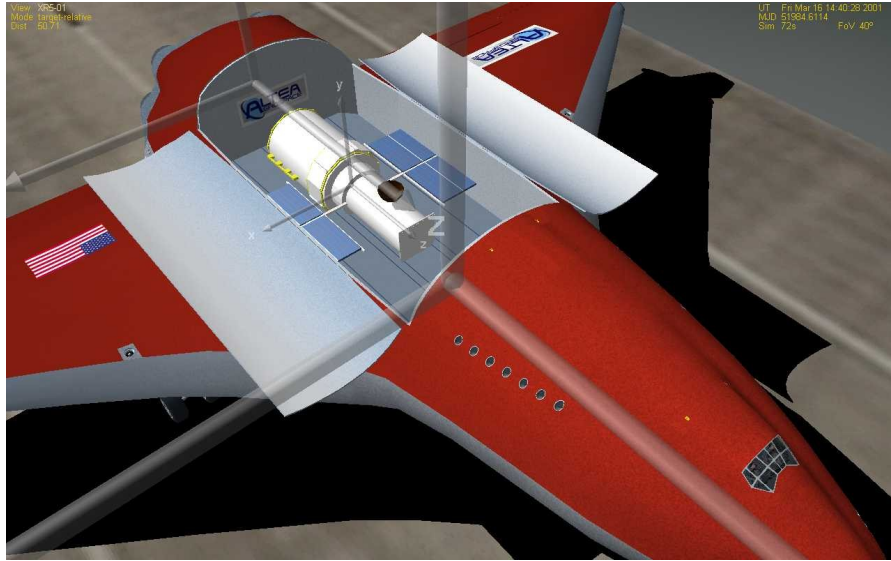


You should notice two things. First, all the interesting visual detail on the mesh (the NASA logo, the attachment points, etc.) are facing downward (just like they were when the vessel was landed). It would be nice to see these things when we have the telescope in the bay. Secondly, if you pan the camera around to the back, you'll see that the Hubble mesh is actually cutting through the floor of the XR5.



Why is this happening? The Hubble's mesh is much taller than one payload slot, as we saw in Step 4, but right now we've defined the attachment point at the vessel's center of mass. Therefore, the centerline of the Hubble is aligned with the centerline of the lower payload slot, and the mesh is overflowing as a result.

To correct this, and the problem of the telescope being upside down, we need to look again at the Hubble's coordinate system:



Payload Orientation

First, we should rotate the Hubble the way we want it. As I mentioned, the NASA logo is on the bottom, which is in the -y direction. This is the direction we want to point up. So, go back to the configuration file and modify the attachment point definition from:

```
P 0 0 0 0 1 0 0 1 XRCARGO
```

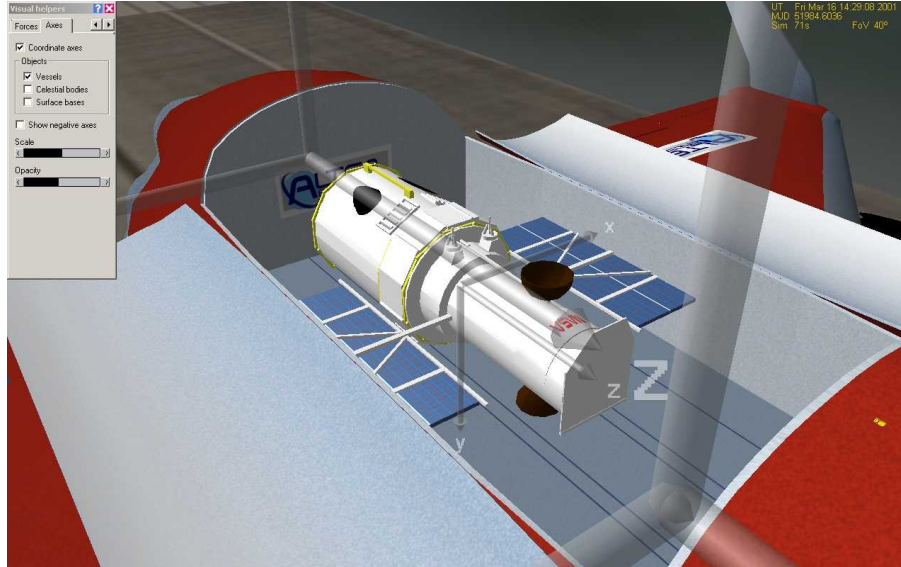
To:

```
P 0 0 0 0 -1 0 0 1 XRCARGO
```

All we've done is change the ROT vector so that the -y direction will point up instead of the +y.

Payload Location

Now, we need to adjust the location of the attachment point so that the Hubble's mesh does not go through the floor of the payload bay. Because the Hubble is taller than one payload slot, we want its attachment point to be located *below* its centerline when it is grappled in the bay. Normally this would mean moving the attachment point in the -y direction. However, **we just flipped the payload upside down in the last step, so -y is now UP, NOT DOWN!** So, moving the attachment point DOWN in the XR5's reference frame means moving it in the +y direction in the Hubble's reference frame.



It takes some trial and error to find exactly the right distance to move the attachment point. You can estimate if you know the size of the payload and can make a guess of how far below the floor the mesh is going. I'm going to guess that moving the Hubble up 1.3 m would solve the problem.

Go back to the configuration file and modify the attachment point definition from:

```
P 0 0 0 0 1 0 0 0 1 XRCARGO
```

To:

```
P 0 1.3 0 0 -1 0 0 0 1 XRCARGO
```

This moves the attachment point 1.3m away from the centerline in the +y direction, which is DOWN when the payload is grappled in the bay.

Step 7: Tweak the Collision Detection for Other Payloads

You are now ready to test the final alignment of your payload in the bay. Launch into the Orbiter scenario again and grapple the Hubble into slot 13. You should see that the NASA logo now faces upward, and the mesh does not go through the floor of the payload bay. You may also notice that the solar panels are just a tiny bit too wide for the bay (they go through the walls) but there's nothing we can do about that – we'll just pretend it fits :-)

Now, I mentioned in Step 4 that by explicitly defining an attachment point for this payload, we bypassed the collision detection code for determining where the Hubble can be grappled. However, *the collision detection code still determines how many slots the payload occupies*. This has implications for how many other payloads can be placed into the bay along with the telescope. If you go to the Payload panel inside the XR5, you should see most of the slots in your bay are greyed out, indicating that they are occupied by the Hubble. This goes back to what we saw in Step 4 – the Hubble is 2.5 slots long, 6 slots wide, and 2.3 slots high. Because the payload bay has 4 slots of length, 5 of width, and 3 of height, we can expect all the payload slots except for the front row to be occupied, on all three levels.

Now, the collision detection system works great for rectangular payloads, but for something more oddly-shaped like the Hubble, it can be helpful to tweak the parameters a bit to obtain a more accurate representation of the space the payload occupies.

The number of occupied slots is determined by two things: the payload's dimensions and the PrimarySlotCenterofMassOffset parameter, which locates the center of your payload relative to the center of its primary payload slot (see schematic figure in chapter 1). **This parameter does not affect where the payload actually sits in the bay, it only affects how many slots the payload manager registers as “occupied.”** As a general rule, this vector should be the reverse of your attachment point location vector. In our case, the location of the attachment point is 1.3m below the centerline of the Hubble (in the XR5 coordinate system), so the PrimarySlotCenterofMassOffset parameter should have a value of 0 1.3 0, meaning that the center of mass is 1.3m ABOVE the attachment point.

Add this parameter to the configuration file, then launch Orbiter and look at how much space the payload takes up in the bay. Compare what you see with the “occupied” slots shown in the XR5 payload panel. Do they correspond well? Sometimes with a strangely-shaped payload, the manager will register slots as occupied that your payload actually doesn't touch, or only just grazes. Depending on how tightly you want to be able to pack the payload bay, you may want to adjust either 1) the size or 2) the offset vector to achieve the desired effect.

In this example, observe that the Hubble doesn't really infringe on the 3rd level slots (or just barely touches them), so I want to reduce the height of the Hubble so that the top row does not register as “occupied.” (Remember, since we explicitly defined the payload slots that are “legal” for the Hubble, the dimensions don't matter for determining where it can be grappled). We can do some simple math to figure out how to adjust the y (height) dimension.

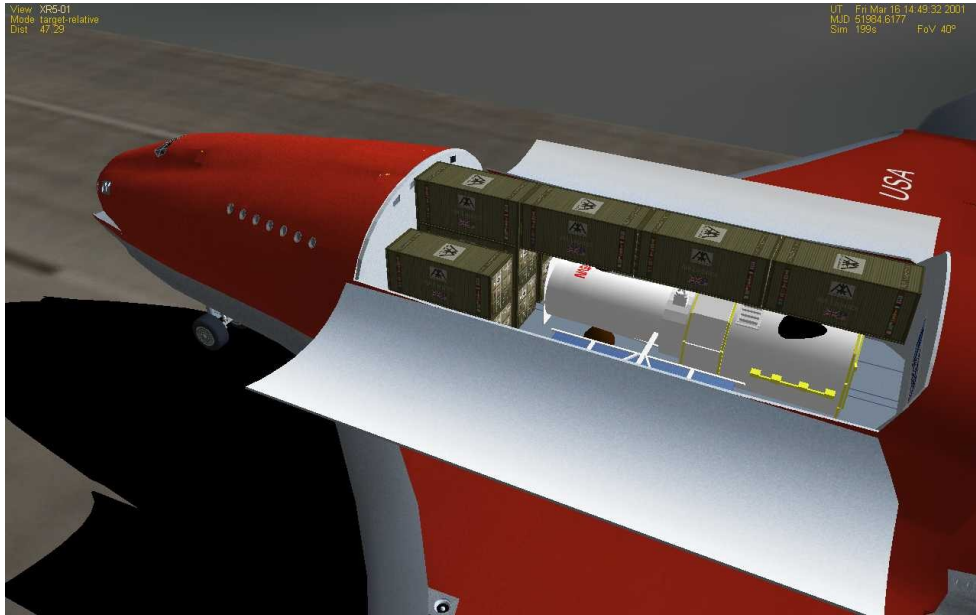
Height of one payload slot = 2.59 m

Height of Hubble center of mass above center of payload slot = 1.3 m (see above)

The vertical distance between the Hubble center of mass and its top (half of its height) must be less than the height of the top of the 2nd row payload slot, or else it will trigger “occupied” status for level 3:

$$\begin{aligned}\text{Max Hubble height} &= (\text{total height of the 2}^{\text{nd}} \text{ level slot}) + (\text{height from the center of 1}^{\text{st}} \text{ level slot to top of 1}^{\text{st}} \text{ row slot}) - (\text{distance from center of 1}^{\text{st}} \text{ row slot to center of Hubble}) \\ &= 2.59 + 2.59/2 - 1.3 \\ &= 2.585 \text{ m}\end{aligned}$$

The total height is then $2 * 2.585 = 5.17$ m. We will round to 5.10. Go back to the Dimensions parameter and change the y value to 5.10. You should now be able to grapple the Hubble into your bay, while leaving the front row and the third level payload slots free:



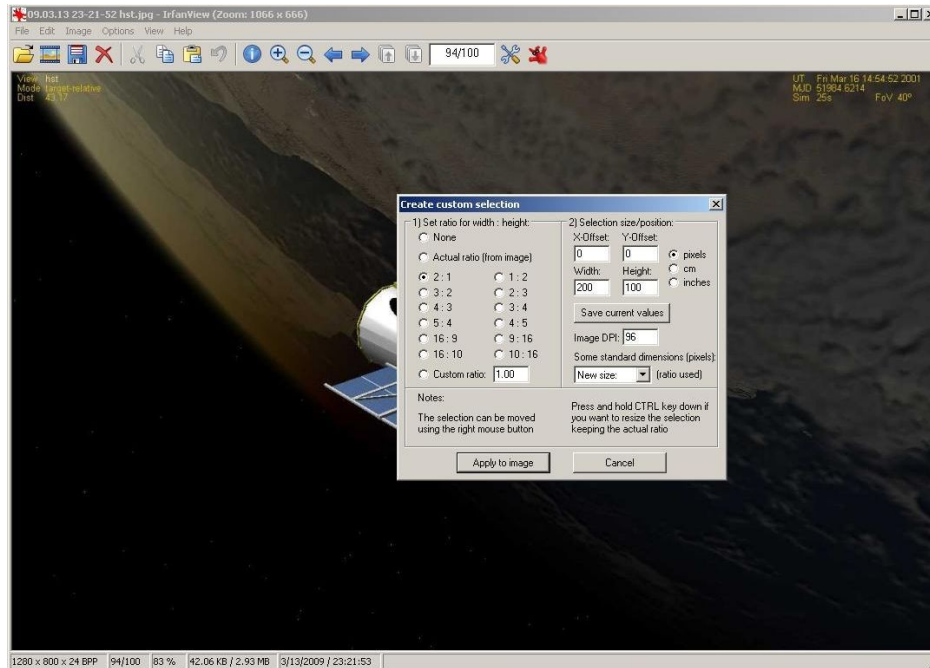
Adjusting the fit of oddly-shaped payloads in this manner is just a matter of taste – how tightly do you want to pack your bay? How much clearance does a payload need? The bottom line is that once you have explicitly defined payload bay slots, you can modify the dimensions to achieve the proper fit without impacting how your payload is grappled.

Step 8: Make A Thumbnail Image

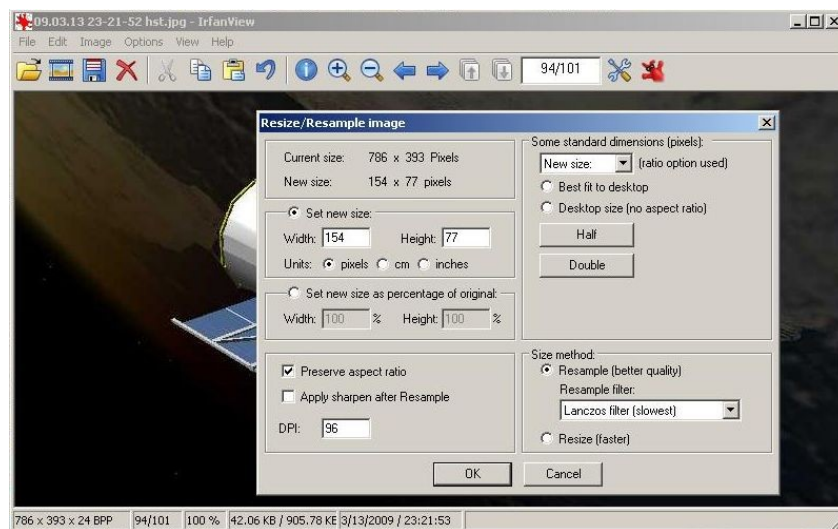
The XR Payload Manager provides the ability to include a thumbnail image with your payload, which is displayed in the Payload Manager window (to make picking it out of the list easier) and in the XR5 / XR2 panels, which makes it easier to identify the correct payload. In my view, **no XR payload is complete unless it includes a screenshot**. While it takes a small amount of extra effort to produce, a screenshot finishes off the payload very nicely and makes it much more usable.

The first step is to take a screenshot of your payload in Orbiter. You can do this with the Print Screen key or with the Screen Capture addon for Orbiter. Next, open your screenshot in Irfanview (or editor of your choice). In order to work with the XR vessels, the screenshot must be saved as a .bmp and have dimensions of 154x77 pixels.

First, crop your image to a 2:1 aspect ratio. In Irfanview, go to Edit > Create Custom Crop Selection > and select “2:1” on the left hand side of the dialog box:



Click “Apply to Image.” You will then see a box with the appropriate aspect ratio that you can move or resize by holding down the right mouse button (move) or the CTRL key (resize while preserving aspect ratio). Adjust the crop region the way you want it, then choose Edit>Crop Selection. Next, resize the image by choosing Image > Resize / Resample. In the dialog box that appears, enter 154 pixels for the width and 77 pixels for the height:



Finally, choose File > Save As and save the image as a 24-bit .bmp file. It is preferable (for the sake of clarity) to use the same name as the configuration file for the payload you are working on (in this case HST.bmp).

Move the file to an appropriate Orbiter directory (such as Orbiter/Config/XR Payload Images/) then open your configuration file and add the following line:

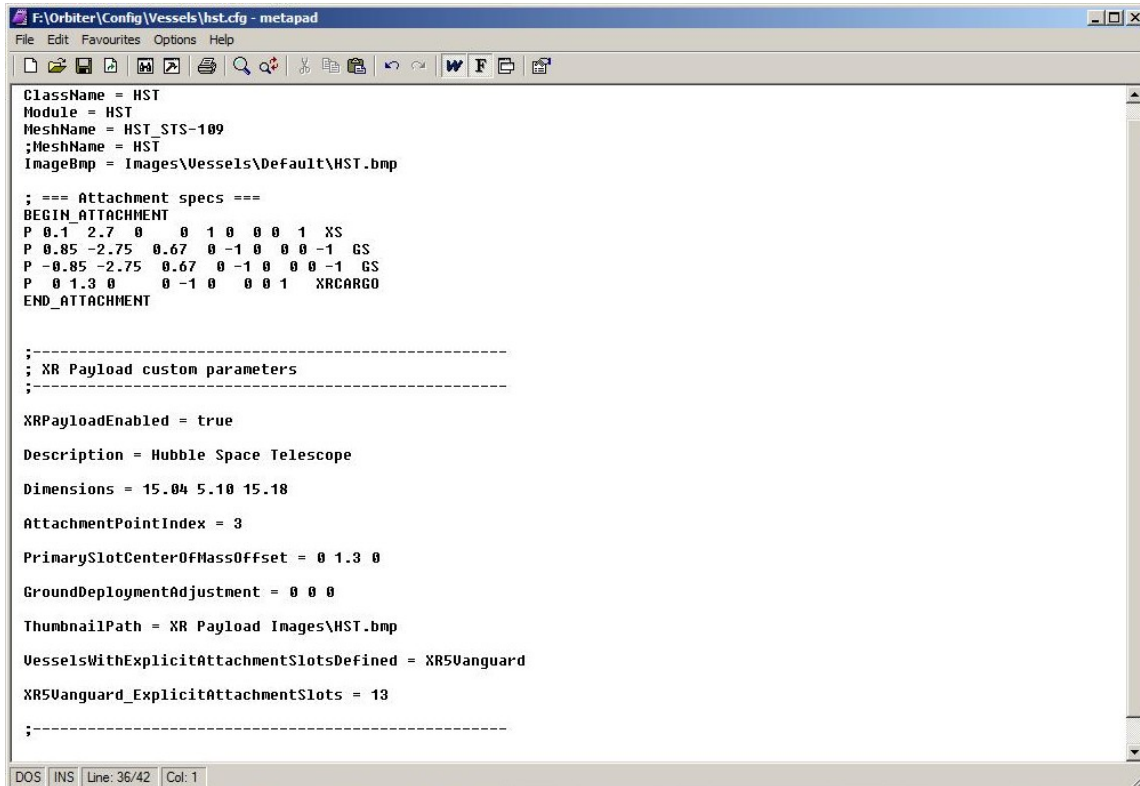
ThumbnailPath = XR Payload Images\HST.bmp

Step 9: Check for Bounce

Proceed exactly as in Step 5 of the Carina tutorial to adjust the ground deployment height and avoid the bounce bug, if needed.

Step 10: Final Test

With everything said and done, your HST.cfg file should look something like this:



```
F:\Orbiter\Config\Vessels\hst.cfg - metapad
File Edit Favourites Options Help

ClassName = HST
Module = HST
MeshName = HST_STS-109
;MeshName = HST
ImageBmp = Images\Vessels\Default\HST.bmp

; === Attachment specs ===
BEGIN_ATTACHMENT
P 0.1 2.7 0 0 1 0 0 0 1 XS
P 0.85 -2.75 0.67 0 -1 0 0 0 -1 GS
P -0.85 -2.75 0.67 0 -1 0 0 0 -1 GS
P 0 1.3 0 0 -1 0 0 0 1 XRCARGO
END_ATTACHMENT

;-----
; XR Payload custom parameters
;-----

XRPayloadEnabled = true

Description = Hubble Space Telescope

Dimensions = 15.04 5.10 15.18

AttachmentPointIndex = 3

PrimarySlotCenterOfMassOffset = 0 1.3 0

GroundDeploymentAdjustment = 0 0 0

ThumbnailPath = XR Payload Images\HST.bmp

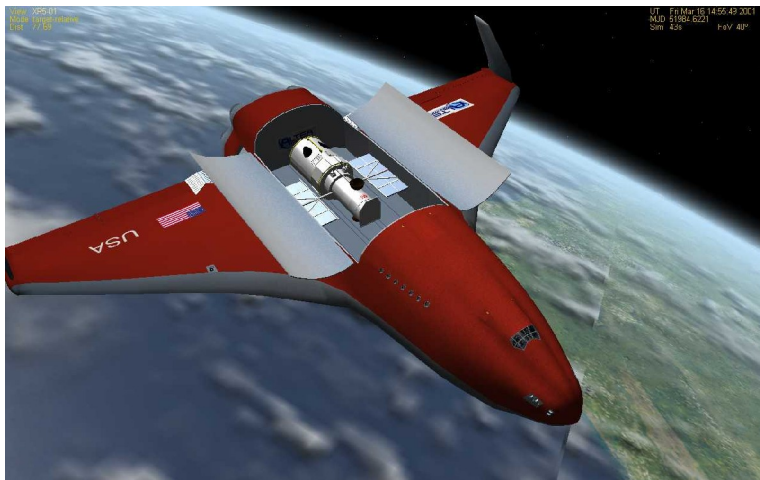
VesselsWithExplicitAttachmentSlotsDefined = XR5Vanguard

XR5Vanguard_ExplicitAttachmentSlots = 13

;-----
DOS INS Line: 36/42 Col: 1
```

Launch into Orbiter and open the Payload Manager (ALT+B) for a final test. When you select “HST” from the payload menu should see your thumbnail image. Then choose “Fill Payload Bay” and the Hubble will be placed in slot 13. You will notice that the front row and the third level of payload slots remain open, while the rest are occupied. You'll also note that the NASA logo is pointing up.

Congratulations! You now have a fully XR-compatible Hubble Space Telescope!



Chapter 4: Notes for Vessels with .dll Modules (including Spacecraft3.dll)

It's absolutely possible to define configuration files for payloads that have their own .dll modules, such as the Dragonfly, the ShuttlePB, or any Spacecraft3.dll vessel. But, sometimes these kinds of payloads introduce some additional complexity in the process because of the way their code is written.

Additional Considerations for Payloads with .dll Modules

1. There may be attachment points defined in the code of the .dll module (rather than in the configuration file. This makes it challenging to figure out the correct AttachmentPointIndex value, because you won't necessarily be able to see all the attachment points. Orbiter assigns attachment points created by the .dll indexes first, THEN begins counting those defined in the configuration file. So, if a payload has two attachments defined in its module and one defined in the configuration, the latter attachment has an index value of 2 (0,1 in the module, 2 in the config file).
2. Depending on the code of the vessel module, spawning / deleting the payload via the XR Payload Management dialog may cause CTD or unexpected behavior. For example, the Dragonfly module can be spawned, but deleting it causes a CTD. The ShuttleA Payload containers can be spawned or deleted, but the meshes are not loaded (they are invisible) until Orbiter is restarted. Be on the lookout for quirks like this, which may have NOTHING to do with the way you've defined the payload configuration.

Special Issues with Spacecraft3.dll

1. For Spacecraft3.dll vessels, all attachment points determined by the module are defined in the vessel's .ini file located in the /Orbiter/Config/Spacecraft/ directory. If you open this .ini file you will see two kinds of attachment points – parent and child points. Spacecraft3.dll has a peculiar way of counting these – child attachment points in the .ini file come first, followed by any attachments defined in the configuration file (such as the XRCARGO one), followed by parent attachments in the .ini file. If you know this, you should be able to figure out the appropriate AttachmentPointIndex value.
2. Spacecraft3.dll vessels cannot be spawned by the Orbiter Scenario Editor or by the XR Payload Management dialog.

Acknowledgements

Many thanks to Dr. Martin Schweiger for creating the fantastic simulator that is Orbiter! Also, thanks to Doug Beachy for the excellent XR-series spacecraft addons.

Disclaimer

You are authorized to use the contents of this pack for personal and non commercial use only. There is no guarantee or support associated with this addon.

Comments / Questions

If you think you have found an error in this document, or find something confusing, please email me at RyanSKingsbury@alumni.utexas.net or post something on Orbiter-Forum.com. I hope you've found this tutorial helpful and will go on to make many payloads for the XR vessels!

Happy Orbiting!

-Ryan Kingsbury ("Countdown84")

Appendix: Quick Reference Guide for Configuration File Parameters

This section provides a complete list of the configuration file parameters available to the XR Payload Management system and gives commentary on the usage of each one. This is intended to expand on the instructions in the example .cfg files included with the XR5 and XR2 and provide a quick-reference to the payload developer. Italics denote optional parameters.

Keep Track of your Reference Frame – A Summary of Vector Parameters for XR Payloads

ALL vectors in the configuration file EXCEPT for the attachment point location must be written with respect to the XR vessel's coordinate system (X = wingtip to wingtip, Y = ground to top, Z = tail to nose). This is especially important to remember if you rotate your payload by setting the (ROT) or (DIR) parameters to values other than (0 1 0) and (0 0 1)!

Parameter (vector)	Coordinate System
Attachment Point (x,y,z) (ROT) (DIR)	Payload
Dimensions (x,y,z)	XR
PrimarySlotCenterOfMassOffset (x,y,z)	XR
GroundDeploymentAdjustment (x,y,z)	XR

Standard Payload Slot Dimensions are 2.4384 wide (X), 2.5908 high (Y), 6.096 long (Z)

-----Parameters Available in an XR Payload Configuration File -----

```
BEGIN_ATTACHMENT
P (X,Y,Z) (ROT) (DIR) XRCARGO
END_ATTACHMENT
```

(X,Y,Z) – location of attachment point in meters, to be grappled into the center of a payload slot.
(ROT) – direction that points up when payload is attached
(DIR) – direction that points forward when payload is attached

- Must be specified relative to the payload's coordinate system
- The flag “XRCARGO” designates the attachment point for the XR payload manager
- The attachment point will be placed at the center of the payload slot
- (ROT) and (DIR) must be perpendicular! If they are not you will get a CTD

XRPayloadEnabled = true

- Required for XR Payload Manager to recognize the vessel as a payload
- Make sure there is only one BEGIN_ATTACHMENT block in the configuration file

Description = ?

Input: a string value of up to 127 characters. 40 or fewer is recommended by Doug.

- The description is not what appears in the list of payloads in the Payload Manager window. That list is populated by the names of the .cfg files themselves.

Dimensions = (X,Y,Z)

Input: dimensions of the payload in meters. X = width, Y = height, Z = length.

- Must be specified relative to the XR vessel's coordinate system. E.g. Z is the length of the payload along the XR5's z axis. If the payload is rotated during attachment this will not necessarily correspond to the Z dimension of the payload itself.

AttachmentPointIndex = n

Input: index of the attachment point to be used when grappling into XR vessels. Numbering starts with 0.

- For vessels with attachment points defined by .dll modules, counting of attachment points defined in the config file begins after counting those defined in the .dll.
- For vessels driven by Spacecraft3.dll, child attachment points defined in the .ini file are counted first, followed by those in the config file, followed by parent attachments in the .ini file.

PrimarySlotCenterOfMassOffset = (X,Y,Z)

Input: Vector describing the distance from the payload's attachment point to its geometric center, in meters.

- Must be specified relative to the XR vessel's coordinate system.
- Does not affect the placement of the payload in the bay; only affects collision detection for grappling and/or determining how many adjacent slots are occupied
- In general, this vector should be the reverse of the attachment point location vector (which points from the geometric center to the attachment point).

ThumbnailPath = ?

Input: path to the payload's thumbnail image

- Path is relative to Orbiter\Config\ directory
- Recommended path is Orbiter\Config\XR Payload Images (Countdown84's recommendation)
- Image must be 154 x 77 pixel 24-bit bitmap.

VesselsWithExplicitAttachmentSlotsDefined = <parent vessel classname>

Input: Names of the modules for vessels for which you want to define explicit attachment points.

- For XR5, use “XR5Vanguard.” For XR2, use “XR2Ravenstar”
- Bypasses the collision detection calculations for determining whether a payload will fit in its assigned slot
- Each <parent vessel classname> should have a matching “<parent vessel classname>_ExplicitAttachmentSlots” line

<parent vessel classname>_ExplicitAttachmentSlots = n1, n2, n3

Input:

n1, n2

numbers of specific payload slots that the payload can be grappled into.

<parent vessel classname> Names of the modules for vessels for which you want to define explicit attachment points. For XR5, use “XR5Vanguard.” For XR2, use “XR2Ravenstar”

- Bypasses the collision detection calculations for determining whether a payload will fit in its assigned slot
- Use the payload management window (ALT+B) to identify identify slots by number

GroundDeploymentAdjustment = (X,Y,Z)

Input: A vector describing the distance to offset the payload from its normal position when deployed to the ground, in meters.

- Must be specified in the XR vessel's coordinate system
- Use this parameter if you observe the bounce bug on deploying a payload to the ground.

PropellantResource1 = x

PropellantResource2 = x

PropellantResource3 = x

Input: quantity of main (1), SCRAM (2), or LOX (3) fuel that a payload holds, in kg.

- Use this parameter to make external fuel tanks for an XR vessel.
-