Foundation Technologies 20050703

Finished by: MLeo Daalder
Originally by: Dasher

0. Contents:

1. What does this mod do?

Foundation Technologies (aka FTech) is designed to revolutionise the way scripters and ship modders can add (new) technologies to BC (and it's ships).
It provides scripters with the basic structures to create new things of which you only need a couple of things to add and you have a new technology and it allows an easy setup for loading your new technology into BC.
And it provides ship modders an easy way to include technologies to their ships.
More on these things later.
It also incorporates Inaccurate Phasers for as well the torpedoes and pulse weapons and for non-player ship.
In the standard package there will be several technologies included of which you will be familiar through ATP.
FTech is more or less designed to be "the next level" for ATP. But all the technologies are plugins.

2. Requirements:

The Foundation (get it with BCMP or BCMI).
Patch is optional.

3. Installation:

Unzip the contents of the zip to your Bridge Commander directory.
And make sure the contents of the scripts directory (from inside the zip) comes in BC's script directory.
And not to forget, make sure that the directory structure stays preserved.

And don't forget that you need to activate the mutator!

4. Included in this package:

Foundation Triggers (by Dasher).
Foundation Technologies version 20050703
Standard Technologies:
- Ablative Armour
- Multivectral Shielding
- Regenerative Shielding
- Reflector Shielding (aka Corbonite Shields)
- Immunity (for disablers)
- Disablers:
  - Power Disabler
  - Sensor Disabler
  - Warp Disabler
  - Impulse Disabler
  - Shield Disabler
  - Cloak Disabler
  - Multiple Disabler
  - Random Disabler
- Timed/Splitting Projectiles:
  - Splitting Single Target
  - Splitting Multiple Target
  - Cloak Disruptor
- Projectiles:
  - Breen Damper

Second Party Add-ons (developed by me (MLeo) during development of FTech):
- Phase Cloak
- Phased Torpedoes
- Isokinetic Torpedoes (from the S4 voyager episode Retrospect)

5. Credits:

Dasher for his Foundation, which makes Modding not a pain in the neck for BC. And for allowing me to finish this great and revolutionary mod! We are sorry to see you go.

Apollo for ATP (FTech could be considered the next evolution of ATP, and many technologies pioneered by Apollo are included in this).
Sneaker for his Inaccurate Phasers (which Dasher originally adapted for FTech and I adapted to include non-players).
Defiant for lot's of testing and many suggestions.

Beta testers:
Defiant,
Mark (Ignis),
Darkthunder (also thanks for making a start writing down how to use various parts!)
The other people who have tested this and whom I have (shamefully) forgotten.
You know who you are you have deepest gratitude for helping this out!

6. How to use:

First of all, any questions can be send to MLeoDaalder@netscape.net.
Second, all code in this document is in Courier New size 10.
Because consistent indentation (so don't mix up 1, 2, 3 or a tab char in the same file) I've done it so that all the code is indented from the beginning of the file, so you could just copy the lines of a code and paste them on a line.

6.1 How to add a technology to a ship:

First thing to know is that FTech uses the ships plugin to determine which technologies a ship uses. This also ensures that if the end user doesn't has FTech, so that he or she won't be experiencing any negative effects from using ships that have Technologies attached to them.

So we add a dTechs part to the ship plugin.
Open the ship plugin and look for a line with Foundation.ShipDef. (note the dot at the end).
Copy that part plus the part before the first space and paste it to the next line. This part is also known as the "ShipDef line of a ship".
Then add ".dTechs = {}" behind it (with out the "").

Any tech you may want to add to the ship will be defined between the "{" and the "}".
The structure between those the brackets, is like this:

```
"Technology Name": Any value specified by the tech,
```

The amount of times you can do that is only limited by the amount of memory in your computer (though there is no reason to overdo it).
I will be referring to this as the Tech Key later in the document.

The "Technology Name" and "Any value specified by the tech" are specific for the Technology you are adding to the ship. Consult their readme's and documentation for more information on those (consult this document for the technologies mentioned above).

If you want to add a technology to a stock ship, you are going to need to look in StaticDefs.py and copy the right ShipDef line to another file with as top “import Foundation” (no “”). You can place the file in Autoload or in Ships. Suggested is that you put it in Autoload so it won’t get disabled the next time you start up BCMP.

A note of interest, FTech is designed so that if it can’t find the tech as listed in the Ship’s plugin file it will give a warning in the console, but it will not impair the ability to use BC or the ship. You will simply not be using the Technology.

6.2 How to add a Technology to a projectile:

Currently one can only add 1 “on fire” and 1 “on yield” technology to a projectile.
The process is the same for Torpedoes and Pulse weapons.
Both use the fire sound to determine what to use.

Normally a modder adds this to the bottom of his/her projectiles script:

```
sYieldName = ''
sFireName = ''

try:
	import FoundationTech
	import “Path to script”

	oFire = “Path to script”.”Name”('Name to describe projectile
tech', {Any Technology specific configuration})
	FoundationTech.dOnFires[__name__] = oFire
	FoundationTech.dYields[__name__] = oFire
except:
	pass
```

But I hope it’s going to be a standard (and I actively urge modders to do so!) to include an example on how to add a projectile technology correctly. I have included examples on the standard projectile technologies.

6.3 Where to install a new Technology:

For this purpose (and to keep Autoload a bit cleaner) I’ve created a new directory called Techs in Custom, if you place those there, then the technologies will get loaded into the system from there.

So unless specified by the readme of the Technology you are trying to install you can usually put the file in Techs.

6.4 How to create Technology X:

Well, it all depends on what you want X to do (I really mean this, lots of things I’m going to tell really depend on what you may want it to do so things are or aren’t needed for what you want it to do).
And there isn’t 1 good answer; there are more than a dozen good answers to fit the idea of what Technology X should/must do.

It usually helps to look at the standard techs that have been supplied with this package. If you really can't find it, contact me and I'll see if I can find it. And if you're a first time scripter, you really ought to know how to use the Console, you're going to need it (even the non first time scripters need it).

But this is how it's generally done:
You start with importing 2 files, App and FoundationTech and of course any other files you may need.

Then you have to create a class that inherits from a "super" class called TechDef. This class can be found in FoundationTech.
So your class definition line may look like this:

```
class TechXDef(FoundationTech.TechDef):
```

The name is arbitrary as long as it uses any letter (abc) number or _ sign and it starts with a letter or the _ sign a class (or function or variable) name may _not_ start with a number.

I won't bother you with explaining the concept of the so-called Object Oriented programming features we are using here, apart that it's a way to use a basic class with all the things you need for a Technology without the need to reinvent the wheel over and over again and it allows you to change the parts that you don't like or you can add parts that you may like to be included without overwriting the super (or base) class.

For projectile-based technologies you usually provide your own
```
def OnFire(self, pEvent, pTorp):
```
and
```
def OnYield(self, pShip, pInstance, pEvent, pTorp):
```
functions based on what you need.
The OnFire function is called when a projectile is launched and OnYield is called when a projectile hit's something.

Here's a run down on what the parameters do:
self is the Object itself, pEvent is the event generated by BC and pTorp is the torpedo in question.

In those functions you put the code that needs to be run for whatever you want it to do.

For non-projectile based technologies there are some more things that you need to do.
First of all, you need to "attach" your Technology to the ships instance. Luckily for you, TechDef already has a function for this and it attaches itself to a variable called lTechs by default. So if you aren't going to use the technology to defend your ship against attacks then you aren't going to be needed to create the following 2 functions:

```
def Attach(self, pInstance):
def Detach(self, pInstance):
```
Those 2 functions attach and remove itself from the lTechs list in pInstance (pInstance is FTechs representation of a ship).

If you are going to react to phasers, pulse weapons, torpedoes or tractor beams you are going to need to override that. But the
`pInstance.lTechs.append(self)`
part must be there in the Attach function.
And:
`pInstance.lTechs.remove(self)`
part must be in the Detach function.

I'll give you the variables you can use:
Phasers: lBeamDefense
Torpedoes: lTorpDefense
Pulse: lPulseDefense
Tractor Beam: lTractorDefense

So if you want your tech to respond to a Phaser blast, you need to add to the Attach function:
`pInstance.lBeamDefense.append(self)`
And to the Detach function:
`pInstance.lBeamDefense.remove(self)`

So it can look like this:
```
def Attach(self, pInstance):
	pInstance.lTechs.append(self)
	pInstance.lBeamDefense.append(self)

def Detach(self, pInstance):
	pInstance.lTechs.remove(self)
	pInstance.lBeamDefense.remove(self)
```
But for each defense list you add the tech to, you will have to give another function. Namely the function FTech calls when a certain event happens.

I'll give you the required functions here:
Phasers:
```
def OnBeamDefense(self, pShip, pInstance, oYield, pEvent):
```
Torpedoes:
```
def OnTorpDefense(self, pShip, pInstance, oYield, pEvent):
```
Pulse:
```
def OnPulseDefense(self, pShip, pInstance, oYield, pEvent):
```
Tractor beam:
```
def OnTractorDefense(self, pShip, pInstance, oYield, pEvent):
```

If any of those functions returns not 0 then the loop, which goes through all the attached technologies (there can be any number of techs attached to a ship), will be broken. The event will be considered "handled".

Of course, you can make a Tech act to multiple events (Ablative Armour is one of them).

Sometimes you also need to get some information about systems on a ship (or even change things on a ship).
For this is purpose, you can overwrite the following functions:

```
def AttachShip(self, pShip, pInstance):
def DetachShip(self, pShip, pInstance):
```

6.5 How to use Disablers in code

Disablers are most known for the Torpedoes that can disable systems. But scripters can use disablers without the need to spawn a torpedo.

The process of doing this is easy.
First of all, the scripter must get the instance of the ship of which the scripter wants to disable a system.

This is done with the following code:

```
pInstance = None
if FoundationTech.dShips.has_key("Name of Ship"):
	pInstance = FoundationTech.dShips["Name of Ship"]
else:
	# Error code here…
	return
```

Here "Name of ship" is the of course the ship you want to disable a system on.
And where you see # Error code here… you can put any code you want for handling if there isn't an instance (for whatever reason, it shouldn't happen but it is possible), like a print statement to the console. Of course the return isn't needed, but then you would need to check if pInstance isn't None.

Ounce you have a valid pInstance (so it isn't None), you can use it for your needs. In this case, to disable a system.

Then you get the system from the ship you want to disable.
Say you want to disable the shields of a ship.
You'd do the following:

```
pInstance.Disable(pShip, pShip.GetShields(),
TimeToDisable, bForce)
```

pShip is the ship itself, you will have to see how you get it.
pShip.GetShields() speaks for itself I think, it gets the shields of a ship.
You can look in App.py under class ShipClass for more of those functions to get systems.
TimeToDisable is a number in seconds that you want the system to be disabled.
bForce is an optional parameter, which you don't have to include.
By default it's 0, but if you set it to non-zero then the code will skip the Disabler Immunity part (so it will force the system to be disabled).

If you (or something else) also disables the same system later on, then the longest duration will be taken.

There also is a function to disable all subsystems of a particular "parent" system (like the Phaser System, it has multiple Phaser banks). This function is "DisableSubSys" and it works exactly the same as Disable.

7. How to use the standard technologies:

For all the ship based technologies I suggest you read 6.1 of this document first.
For all projectile-based technologies I suggest you read 6.2 of this document first.

Make sure you understand it before continuing.

7.1 Ablative Armour

The Ablative Armour protects your ship until the armour is destroyed.

The Ablative Armours Technology Name is: "Ablative Armour".
The configuration is the number of hit points the armour has.
It isn't needed for the ship to have a non-primary hull property (optionally target table and critical), though if you want the health bar to display the status of the Armour, you will need a hull property called "Ablative Armour" and with the same amount of hit points you define in the dTechs key.

Your dTechs key for this tech could be something like this:

```
"Ablative Armour": 6000,
```

"Ablative Armour" is the default name for the hull property, if you want to give a different name to it, you can do that by adding "[" and "]" around the amount, and adding a new name behind it.
Your dTechs key will look something like this then:

```
"Ablative Armour": [6000, "Hull Plating"],
```

7.2 Multivectral Shielding

Multivectral Shielding makes all your shield vectors work as 1 shield emitter. So damage is distributed across all sides (it is advised that you give each part the same maximum charge and recharge rate).

The Multivectral Shielding Technology Name is:
"Multivectral Shields".
The configuration is the percent chance that it the technology will succeed.

Your dTechs key for this tech could look like this:

```
"Multivectral Shields": 50,
```

7.3 Regenerative Shielding

Regenerative Shielding sends some of the damage done to your shields back into your shields. So it uses part of the damage to reinforce it self.

The Regenerative Shielding Technology Name is:
"Regenerative Shields".
The configuration is a number of which the damage done to the shield is divided by (so don't make it 0).

Your dTechs key for this tech could be like this:

```
"Regenerative Shields": 10,
```

7.4 Reflector Shielding (aka Corbonite Shields)

Reflector Shielding will send torpedoes back where they came and pulse weapons flying.
But reflecting puts a mild strain on your shields. Which is 10% of the damage done by the torpedo or pulse weapon.

The Reflector Shielding Technology Name is:
"Reflector Shields".
The configuration is the percent chance that the projectile will get deflected.

Your dTechs key could look like this:

```
"Reflector Shields": 75,
```

7.5 Immunity (for disablers)

FTech allows scripters to easily disable a subsystem on a ship for a specified time without all the hassle of keeping track of everything yourself.
And of course, one can make a ship immune for a certain disabler.
One can even specify how much disabling a system can take.

The Technology Name for this is: "Disable Immunity".
The configuration is a dictionary (a structure like the dTechs like structure with the "{" and "}" sign) containing the systems the ship has a certain level of immunity for.

So your dTechs key could look like this:

```
"Disable Immunity": {Immunities}
```

Where Immunities is one (or more) of the following:

```
"Power":    MaxTimePrevented
"Sensor":   MaxTimePrevented
"Cloak":    MaxTimePrevented
"Shield":   MaxTimePrevented
"Warp":     MaxTimePrevented
"Impulse":  MaxTimePrevented
"Phaser":   MaxTimePrevented
"Torpedo":  MaxTimePrevented
"Pulse":    MaxTimePrevented
```

where MaxTimePrevented is a time in seconds that is the maximum amount of time the ship can prevent from getting disabled, anything larger, and it will be disabled for the remaining amount of time.

A ship can have multiple immunities (and different maximum time per immunity) they are added by appending a "," between 2 immunities (without the " signs)
It becomes like this (just an example):
```
“Disable Immunity”: {“Power”: 15, “Warp”: 1, “Cloak”: 10}
```
As you can see, order doesn't matter.

When MaxTimePrevented is 0 then disablement will always be prevented.

7.6 Breen Damper (aka Breen Drainer Weapon)

The Breen Damper is projectile technology.
It will disable the power producing systems, shields and cloaking device on impact.
It also has an immunity tech (separate from 7.5 because it’s a different system of disabling).

You can add the Breen Damper to your projectile by adding the following code to the bottom of your projectile script:

```
sYieldName = ‘Damper Weapon’
sFireName = None

try:
	import FoundationTech
	try:
		oFire = FoundationTech.oTechs[sFireName]
		FoundationTech.dOnFires[__name__] = oFire
	except:
		pass

	try:
		oYield = FoundationTech.oTechs[sYieldName]
		FoundationTech.dYields[__name__] = oYield
	except:
		pass
except:
	pass
```

The Damper Immunity Technology Name is: “Drainer Immune”.
The configuration is not used so just a 0 or 1 (I admit I prefer 1, it’s more natural to put it there than 0, which could suggest that it wouldn’t be immune) is all that is needed.

Your dTechs key could look like this:
```
“Drainer Immune”: 1,
```

7.7 Cloak Disruptor

The cloak disruptor allows you to disable the cloaking devices of cloaked ships (both enemy and friendly) in the area (in a range of a

1000 game units from the projectile position) by firing a projectile. It will begin disruption after 1 second in flight.
It is possible that a projectile can do multiple bursts.

There is nothing configure about this, so I'll just give you the example:

```
try:
	import FoundationTech
	import ftb.Tech.CloakDisruptor

	oFire = ftb.Tech.CloakDisruptor.CloakDisruptor('Cloak
	Disruptor', {})
	FoundationTech.dOnFires[__name__] = oFire
except:
	pass
```

7.8 Disablers

Disablers are technologies that can disable systems for a specified amount of time. The disablers I mention here are using the Disablers I mentioned at 6.5 and are Projectile based systems.

There are multiple predefined disablers for this already for you to use (you can of course configure or even create one yourself).
This is the basic structure for adding a Disabler to a projectile:

```
sYieldName = 'Disabler Name'
sFireName = None

try:
	import FoundationTech
	try:
		oFire = FoundationTech.oTechs[sFireName]
		FoundationTech.dOnFires[__name__] = oFire
	except:
		pass

	try:
		oYield = FoundationTech.oTechs[sYieldName]
		FoundationTech.dYields[__name__] = oYield
	except:
		pass
except:
	pass
```

'Disabler Name' is of course the technology name.
Here are a couple of predefine ones:

```
'Power Disable'
'Sensor Disable'
'Warp Disable'
'Impulse Disable'
'Shield Disable'
'Cloak Disable'
'Multiple Disable'
'Random Disable'
```

They all disable systems for 10 seconds.

Multiple Disable disables Shields and Sensors for 10 seconds each.

Random Disable randomly picks systems to disable for 10 seconds.
The systems it tries to disable are:

Sensors	(20% chance)
Warp	(20% chance)
Impulse	(20% chance)
Shields	(20% chance)
Cloak	(10% chance)
Power	(10% chance)

You can of course, configure the predefined technologies.
The basic structure changes a little bit because you need to import and create a new instance of one of the predefined disablers.

```
try:
	import FoundationTech
	from ftb.Tech.DisablerYields import *
	try:
		oYield = ClassName('New Name', {Configuration})
		FoundationTech.dYields[__name__] = oYield
	except:
		pass
except:
	pass
```

Here ClassName is one of the following:
PowerDisableDef
SensorDisableDef
WarpDisableDef
ImpulseDisableDef
ShieldDisableDef
CloakDisableDef
MultipleDisableDef
RandomDisableDef

You will have to specify a new unique name for 'New Name'.
Configuration is in the following form for all technologies (apart for MultipleDisableDef and RandomDisableDef):

```
{'tDisable': time}
```

Here time is the time in seconds you want the system (I think the names of the classes speak for themselves) to be disabled.

Here is the configuration for MultipleDisableDef:

```
{'lYields':
	(Disabler, time),
	(Another Disabler, time))
}
```

Time is the same as always, but you can define different times for each disabler.
Disabler and Another Disabler can be one of the following:

```
oPowerDisable
oSensorDisable
oWarpDisable
oImpulseDisable
oShieldDisable
oCloakDisable
oMultipleDisable
oRandomDisable
```

These are all the predefined Disablers (so they have the same effect as I mentioned before). Naturally, you can enter your own.

And not to forget, you can add as many disablers to a MultipleDisableDef, separate them (as seen in the example) with a ‘,’.

And here is the configuration for RandomDisableDef:

```
{‘lYields’:
      (Disabler, time, chance),
      (Another Disabler, time, chance))
}
```

As you can see it’s the same as MultipleDisableDef apart for that you can specify the chance.

7.9 Splitting Projectiles

There are 2 types of splitting projectiles, single target and multi target. Single target splitting projectiles split and converge at the same target. Multi target splitting projectiles split and pick their own (enemy) targets. And both do this after 100 game units distance from the parent ship.

With splitting torpedoes it’s important that you have 2 projectiles.
1 is the “shell” and the other the “warhead”.
The shell is what gets fired from the ship. The warhead is what comes out of the shell. So it’s important that the warhead and the shell aren’t the same, or you would get infinite (well until BC can’t handle it anymore) torpedoes on the screen. This is something you don’t want to be happening.

It is of course all right to have multiple stage cluster projectiles, just try and prevent a loop of any sort with these.

And here is the example for a Single Target Splitting Projectile:

```
sYieldName = ‘’
try:
      import FoundationTech
      import ftb.Tech.TimedTorpedoes

      oFire = ftb.Tech.TimedTorpedoes.MIRVSingleTargetTorpedo(
      ‘Name for tech’, {
            ‘spreadNumber’:   3,
            ‘spreadDensity’:  6.5,
            ‘warheadModule’:  ‘PathToModule’,
            ‘shellLive’:      0,
```

```
	})
	FoundationTech.dOnFires[__name__] = oFire
	oYield = FoundationTech.oTechs[sYieldName]
	FoundationTech.dYields[__name__] = oYield
except:
	pass
```

And here is a quick rundown. Name for Tech is the name you give to this configuration.
spreadNumber is the amount of warheads the projectile will fire.
spreadDensity is the distance between 2 warheads when they spread out from the shell (after which they will converge on their target). It's defined in degree.
warheadModule is the path to the new projectile that is to be spawned (same structure of path as with the projectiles with the torpedoes in the Hardpoint of a ship).

Multiple Target Splitting Projectiles are done the same way.
With 1 difference, 'MIRVSingleTargetTorpedo' will be:
'MIRVMultiTargetTorpedo'.

8. How to use Second Party technologies

This of course doesn't differ from using standard technologies.
I do implore that you read the readme's supplied with the technologies with great care.

And I really implore scripters to include readme's with their technologies.

8.1 Phase Cloak

The Phase Cloak allows ships to pass through other ships and stations without causing collisions (weapons will hit you). Planets and suns are too heavy to pass through (and is rather boring, it's an empty shell, I can know it, I've been there).

The Phase Cloak Technology Name is: "Phase Cloak".
The configuration is a number that defines the amount of time (in seconds) that the cloak will be disabled for when hit by a weapon.

If you put as configuration 0 then it will never be disabled.

Your dTechs key could look like this:

```
"Phase Cloak": 3,
```

8.2 Phased Torpedoes

Phased Torpedoes (or Projectiles) will go right through the shields of a ship and collide with the ship itself. So it won't do hull damage.

This mod is created so that it will add the Phased Plasma from Episode 7 and 8 from the Single Player are added without further configuration.

Here is how you add it to your projectile:

```
try:
	sYieldName = 'Phased Torpedo'
	import FoundationTech
	try:
		oYield = FoundationTech.oTechs[sYieldName]
		FoundationTech.dYields[__name__] = oYield
	except:
		pass
except:
	pass
```

8.3 Isokinetic Torpedoes

These are the projectiles from the Voyager Season 4 episode Retrospect. They are so damaging that they continue even after the shields have collapsed.

(A note for scripters: Yes, this is a failed Phased Torpedo script. ☺)

Here is how you add it to your projectiles:

```
try:
	sYieldName = 'Isokinetic Cannon Round'
	import FoundationTech
	try:
		oYield = FoundationTech.oTechs[sYieldName]
		FoundationTech.dYields[__name__] = oYield
	except:
		pass
except:
	pass
```

And a note, there have been bugs encountered with this script. Random torpedoes flying after the shields have collapsed.

Thank you for reading this readme and how to guide to Foundation Tech!
Any questions can be send to MLeoDaalder@netscape.net
Or contact me through PM on Bridge Commander Universe.