Help me file – Fault tolerant planning convertor

1. Property file :

   The Convertor application accepts parameters from a file. The parameter file uses an XML format. The top level element in the parameters file is named *parameters*. Each parameter is defined by an entry element. An entry element has an attribute "key" which defines the parameter.

- **K** – number of possible faults in the domain
- **domName** – the name of the domain
- **probName** – the problem name
- **domain** – the directory of the XML input file, defines the domain
- **problem** – the directory of the XML input file, defines the problem
- **domOutput** – the output directory for the domain pddl file the convertor creates
- **probOutput** – the output directory for the problem pddl file the convertor creates

An example for property file :

```
<properties>
<comment>Ron and Mor FT convertor Settings</comment>
<entry key="k">0</entry>
<entry key="domName">8-Puzzle</entry>
<entry key="probName">8-Puzzle_3X3</entry>
<entry key="domain">./ProbFiles/8-puzzle_domain.xml</entry>
<entry key="problem">./ProbFiles/8-puzzle_problem.xml</entry>
<entry key="domOutput">./ProbFiles/8PuzzleDom.pddl</entry>
<entry key="probOutput">./ProbFiles/8PuzzleProb.pddl</entry>
</properties>
```

2. Domain XML :

   The Domain Input file, is a non-deterministic planning domain, represented by xml tags :

Xml tag	Description
`<Domain>`	Open tag
`<Name>8-Puzzel</Name>`	Domain name

`<ObjectTypes>`
`<string>position</string>`
`<string>tile</string>`
`</ObjectTypes>`
	Object types used in the domain
	PDDL : (:types position tile levelType)

`<NotDependedPredicates>`
`<Literal>`
`<Name>increase</Name>`
`<Type>Pos</Type>`
`<Params>`
`<Parameter>`
`<Name>?pos1</Name>`
`<IsTypedParam>true</IsTypedParam>`
`<ParamType>position</ParamType>`
	Independent predicates – the same for all the different levels. Won't be copy for each level by the convertor
	PDDL :
	(increase ?pos1 - position ?pos2 - position)

`</Parameter>`
`<Parameter>`
`<Name>?pos2</Name>`
`<IsTypedParam>true</IsTypedParam>`
`<ParamType>position</ParamType>`
`</Parameter>`
`</Params>`
`</Literal>`
`</NotDependedPredicates>`	

`<Predicates>`
`<Literal> …`
`</Literal>`
`</Predicates>`
	Predicates that can have different values for each level / branch.
	Will be copied to each level

`<Actions>`
`<Action>`
`<Name>grab</Name>`
`<Params/>`
`<PreConditions>`
`<Literal>`
`<Name>holding</Name>`
`<Type>Neg</Type>`
`<Params/>`
`</Literal>`
`</PreConditions>`
`<Effects>`
`<Effect>`
`<Name>e0</Name>`
`<EffType>Single</EffType>`
`<AddedLiterals>`
`<Literal>`
`<Name>holding</Name>`
`<Type>Pos</Type>`
`<Params/>`
`</Literal>`
`</AddedLiterals>`
`<DeletedLiterals/>`
`</Effect>`
`</Effects>`
`</Action>`
`</Actions>`
	Action - action with no non-deterministic effects.
	PDDL –
	(:action grab
	:parameters ()
	:precondition
	(and
	(not-in-break-in)
	(not (holding level_0_0))
	)
	:effect
	(holding level_0_0)
	)

`<NonDetAction>`
`<Name>break-in</Name>`
`<Params> …`
`<PreConditions> …`
`<Effects>`
`<Effect>`
`<Name>Deterministic</Name>`
`<EffType>Single</EffType>`
`<MultiParams/>`
`<AddedLiterals> …`
`<DeletedLiterals> …`
`</Effect>`
`<NonDetEffects>`
`<F__Effect>`
`<Name>e0</Name>`
`<EffType>Single</EffType>`
`<MultiParams/>`
`<AddedLiterals/>`
`<DeletedLiterals/>`
`</F__Effect>`
`<F__Effect>`
`<Name>e1</Name>`
`<EffType>Single</EffType>`
`<MultiParams/>`
`<AddedLiterals/>`
	Non deterministic action -
	Non deterministic action Has non-deterministic effects.
	Each effect has an F function .
	PDDL –
	(:action break-in
	:parameters…
	:precondition…
	
	:effect
	(and (not (holding))
	(has ?bowl ?eggs-after)
	(not (has ?bowl ?eggs-before))
	(nondet (not (unspoiled ?bowl))))
	)

<DeletedLiterals> <Literal> <Name>unspoiled</Name> <Type>Pos</Type> <Params> <string>?bowl</string> </Params> </Literal> </DeletedLiterals> <EffectFFunc>1</EffectFFunc> </F__Effect> </NonDetEffects>	

3. Problem XML:

   The Problem Input file, is the non-deterministic planning problem represented by xml tags :

Xml tag	Description
<Problem>	Open tag
<Name>omlette-3</Name> <DomainName>Omlette</DomainName>	Problem name Domain name
<Objects> <string>bowl1</string> <string>bowl2</string> <string>n0</string> <string>n2</string> <string>n1</string> <string>n3</string> </Objects>	Problem Objects PDDL – (:objects bowl1 bowl2 n0 n1 n2 n3)
<InitState> <Literal> <Name>has</Name> <Type>Pos</Type> <Params> <Parameter> <Name>bowl1</Name> <IsTypedParam>false</IsTypedParam> <ParamType></ParamType> </Parameter> <Parameter> <Name>n0</Name> <IsTypedParam>false</IsTypedParam> <ParamType></ParamType> </Parameter> </Params> </Literal> … </InitState>	Initial state PDDL – (:init (unspoiled bowl1) (unspoiled bowl2) (has bowl1 n0) (has bowl2 n0) …)
<GoalState> <Literal> <Name>has</Name> <Type>Pos</Type> <Params> <Parameter> <Name>bowl2</Name> <IsTypedParam>false</IsTypedParam> <ParamType></ParamType> </Parameter> <Parameter> <Name>n3</Name> <IsTypedParam>false</IsTypedParam>	Goal state PDDL- (:goal (and (unspoiled bowl2) (has bowl2 n3))

<ParamType></ParamType> </Parameter> </Params> </Literal> … </GoalState>	

4. Converting from Fault tolerant planning to classic planning -

   After building a property file, domain xml file and problem xml files, we can run the convertor. The convertor is a java application.

   Command for running the Convertor application:

   Java -cp FTcompiler.jar Main.FTPlaningCompMain property_file

   The command first argument should be the property file.