INNOVATION TODAY
FOR TOMORROW®

App Note – Parsing and Creating Configuration Files

nutaq
.com

Revision history

Revision	Date	Comments
1.0	October 2015	First edition.

Table of Contents

1 Introduction

The BAS software suite provides a method for conveniently passing a potentially large number of parameters to a user application. Instead of using command line arguments, it is possible to use what the BAS software suite refers to as Configuration Files, with extension ".ini". Section 2 demonstrates how Configuration Files are structured. The user application then parses the file and retrieves pertinent parameters. Section 0 explains how to use functions provided with the BAS software suite in user applications to parse Configuration Files.

Lots of examples provided with the BAS software suite make use of Configuration Files. For example, see RTDEx and Record/Playback examples in *bas\examples\rtdex_recplay\host\scripts*. Those examples use four configuration files:

- *RecplayTestPlayback.ini*
- *RecplayTestRecord.ini*
- *RtdexTestDownlink.ini*
- *RtdexTestUplink.ini*

See the rest of this document for details, as well as the Record Playback Examples Guide in *bas\doc\cores\RecordPlayback* for specific details on RTDEx and Record/Playback examples and these four Configuration Files.

2 Structure and Content of Configuration Files

Nutaq's configuration files are mere lists of parameters, with very simple syntax. Configuration files follow the rules stated below.

1. Parameters are arranged one by line, each one followed by an equal sign ('=') and the parameter's value following it.
2. Lines beginning with symbol '#' or '**;**' are comment lines and are ignored by Nutaq's parsing functions.
3. Parameter values may be strings, signed integers or unsigned integers. See rule 4 for an example showing different types of parameter values.
4. Parameters may be grouped in sections. A section begins with a name contained within square brackets '[name]' and ends when the next name is found, or when the end of the file is reached. This is an example showing two sections:

   ```
   [FMC1]
   #This is the FMC type in this slot position
   type=mo1000
   active_flag=1
   sampling_clock_source=1

   [FMC2]
   #This is the FMC type in this slot/position
   type=mi125
   active_flag=1
   sampling_clock_source=2
   ```

   First section name is made of single identifier `FMC1` while second name is made of single identifier FMC2.

   Referring to rule 3, parameter `type` has string value "`mo1000`" while parameter `active_flag` has unsigned (or signed) integer value 1.
5. A section name may have more than one identifier. If more than one identifier is used, they are separated by commas with no use of a white space. For example, this section has identifiers `FMC1` and FMC2:

   ```
   [FMC1,FMC2]
   type=mi125
   ```

   This, for example, is not valid, because there is a white space after the comma:

   ```
   [FMC1, FMC2]
   type=mi125
   ```
6. No rule exists as to whether group parameters within a single section using multiple identifiers or spread them across multiple sections using a single identifier. For example, this

```
[FMC1]
active_flag=1
[FMC2]
active_flag=1
```

is equivalent to this

```
[FMC1,FMC2]
active_flag=1
```

In both cases, Nutaq's parsing functions will yield the same result.

Even though the first method leads to longer configuration files, it is sometimes preferred. In the present case, it is the way to toggle parameter `active_flag` for individual FMCs. Sometimes it also leads to clearer configuration files.

7. Configuration files are sensitive on white spaces and case. No white space may exist before and after identifiers, before and after equal signs, etc. Empty lines may however exist between parameters.

2.1 Notes on Typical Configuration Files Usage in BAS Software Suite

The BAS software suite provides utility applications used to configure and enable all FMCs and various FPGA cores. They serve as applicative examples of Nutaq's EAPI as well as general purpose tools. Project folders for those applications reside in the *bas\tools\apps* directory, while pre-compiled versions of those applications are available in the *bas\tools\bin* directory. More details regarding those applications are available in the Programmer's Reference Guide related to the FMC or core that application is designed for. More precisely, see

- *bas\doc\cores* and
- *bas\doc\fmc*

Here is a typical example of a configuration file. In this case, the configuration file contains parameters for two MI125:

```
#This is the FMC type in this slot position
[FMC1,FMC2]
type=mi125

#This is the flag to indicate if the FMC will be initialized or not
#Bypass Initialization = 0
#Initialize FMC = 1
[FMC1]
active_flag=1
[FMC2]
active_flag=1

#This is the sampling clock source
#MI125_CLKSRC125MHZ                 = 0,        ///< Internal 125 MHz (default)
#MI125_CLKSRCEXT                    = 1,        ///< External clock
```

```
#MI125_CLKSRCBOTTOMFMC              = 2,        ///< This setting must not be used for
bottom FMC (main) card, clock top board from bottom board

#MI125_CLKSRCFMCCARRIER             = 4,        ///< FMC carrier clock (future)

[FMC1]

sampling_clock_source=0

[FMC2]

sampling_clock_source=2


#This is the trigger output IO to be disconnected (default) or connected from FPGA to
outside.

[FMC1,FMC2]

trigger_out=OFF
```

All FMC utility applications (named *<FMC>_Init* in the *bas\tools\apps\fmc* directory), use configuration files for passing parameters. They all treat configuration files the same way and expect the same structure. Mainly, configuration files for Nutaq's FMC utility applications adhere to all rules stated in the main paragraph of section 2, with the following additional rules:

8. They all use the same four identifiers in their section names. These section identifiers refer to which FMC in the system is to be configured using parameters defined in the section. They are

    - FMC1
    - FMC2
    - FMC3
    - FMC4

    and they refer to FMCs physical positions in the system. Figure 2-1 demonstrates how identifiers map to positions if using a non-stackable FMC (MI250, ADC5000 and ADAC250). In these cases, identifiers FMC3 and FMC4 are non-applicable and shouldn't be used.

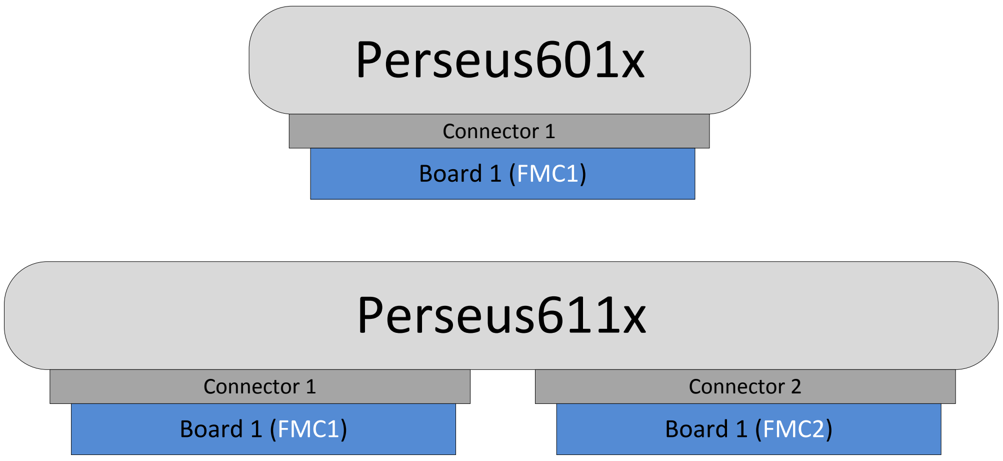

Figure 2-1. FMC identifiers for non-stackable boards

Figure 2-2, demonstrates how identifiers map to positions if using a stackable board (MI125, Radio420, and MO1000).

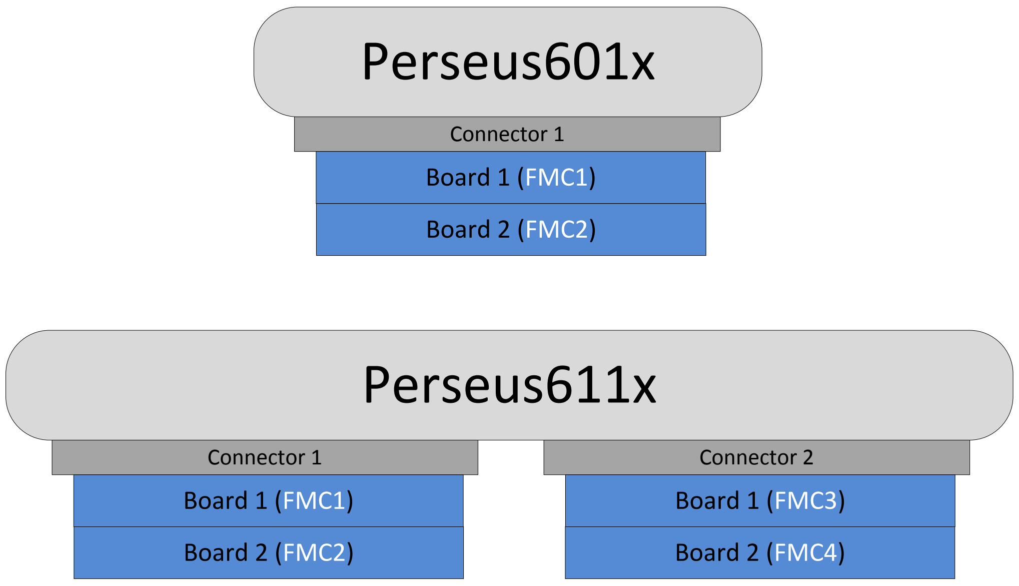

Figure 2-2. FMC identifiers for stackable boards

Please note that the schematic to use depends on the type of FMC installed on system, and not if 2-board stacks are actually present in the system.

9. *<FMC>_Init* will only fetch a parameter when it's in a section related to the FMC that application is designed for, where *<FMC>* is one of six Nutaq's FMC, that is:

    - MI125
    - Radio420
    - MO1000
    - ADAC250
    - MI250
    - ADC5000

    For a configuration file section to be related to a particular FMC, the following rules must be followed,

    - It must have in its name one or more of the four FMC positional identifiers stated in rule 8 (that is `FMC1`, `FMC2`, `FMC3` or `FMC4`).
    - Then, another section, or the present section, must have parameter and value "`type=<FMC>`" defined (again, where `<FMC>` is one of the six Nutaq's FMC).

      If "`type=<FMC>`" is declared in another section, the name of that section must use the same FMC positional identifier as the one used for naming the present section.

    For example, In the example configuration file showed above, "`type=mi125`" is found in a section of name `[FMC1,FMC2]`. So every section having either identifier `FMC1` or identifier `FMC2` in its name, or both, is a MI125 section. *MI125_Init* will then fetch MI125 parameters in those sections.

    In this example configuration file, all sections are MI125 sections. That is because all sections use either identifier `FMC1` or identifier `FMC2` in their name, and `type=mi125` was declared in section with name using both those identifiers.

In the next example, however, FMC3 is linked to a MO1000 FMC, and parameter `active_flag` wouldn't be fetched by application *MI125_Init*.

```
[FMC3]
type=mo1000
[FMC3]
active_flag=1
```

10. Multiple FMC types can be configured in the same configuration file. FMC types and related sections are resolved with rule 9.

    This means that multiple applications may use the same configuration file.

    An example of this can be seen in file *bas\examples\mo1000_mi125\host\scripts\MOMI_Init.ini*:

```
[FMC1]
#This is the FMC type in this slot position
type=mo1000

#This is the flag to indicate if the FMC will be initialize or not
#Bypass Initialization = 0
#Initialize FMC = 1
active_flag=1

#This is the sampling clock source
#eMo1000ClkSrc125mhz                = 1,        ///< Internal 125 MHz (default)
#eMo1000ClkSrcExt                   = 3,        ///< External clock
#eMo1000ClkSrcBottomfmc             = 2,        ///< This setting must not be used
for bottom FMC (main) card, clock top board from bottom board (mandatory for clock
slave)
#eMo1000ClkSrcFmccarrier2           = 0,        ///< FMC carrier CLK2 clock
#eMo1000ClkSrcFmccarrier3           = 4,        ///< FMC carrier CLK3 clock
sampling_clock_source=1

#...
#...
#...

[FMC2]
#This is the FMC type in this slot/position
type=mi125

#This is the flag to indicate if the FMC will be initialize or not
#Bypass Initialization = 0
#Initialize FMC = 1
active_flag=1

#This is the sampling clock source
```

```
#MI125_CLKSRC125MHZ                  = 0,        ///< Internal 125 MHz (default)

#MI125_CLKSRCEXT                     = 1,        ///< External clock

#MI125_CLKSRCBOTTOMFMC               = 2,        ///< This setting must not be used
for bottom FMC (main) card, clock top board from bottom board

#MI125_CLKSRCFMCCARRIER              = 4,        ///< FMC carrier clock (future)

sampling_clock_source=2


#This is the trigger output IO to be disconnected (default) or connected from FPGA
to outside.

trigger_out=OFF
```

In this example, the same file configures two FMCs of different types. Section `[FMC1]` configures a MO1000, and section `[FMC2]` configures a MI125. *MI125_Init* would only fetch parameters of section `[FMC2]`.

3 Configuration Files Parsing Functions

The BAS software suite provides various examples where configuration files are used. Most of the time, configuration files are parsed in FMC utility applications (named *<FMC>_Init* in the *bas\tools\apps\fmc* directory). For parsing configuration files, those applications use two functions defined in file ConfigFile.c, residing in *bas\sdk\host\api\utils\src*. The header declaring those two functions is available under the name ConfigFile.h, in *bas\sdk\host\api\utils\inc*.

3.1 Function List and Usage

Result ConfigFile_GetParamValue(const char * Filename, const char * Section, const char * Parameter, const char * ValueType, void * pValue, uint32_t ValueMaxSize);

Gets the value of parameter *Parameter* and stores it in variable *pValue*. The parameter is read in file Filename in section having name Section.

Parameters

In	**Filename**	Name of file to parse
In	**Section**	Section name in which to look for parameter *Parameter*. If NULL is passed, the function only fetch the value if the parameter isn't placed in any section.
In	**Parameter**	String holding the parameter name. The exact same string is expected to be found in the configuration file.
In	**ValueType**	Format specifier of the parameter to fetch. This is the subsequence beginning with % that is used in standard C functions scanf(), printf(), etc
Out	**pValue**	Pointer to the variable that will hold the fetched value. If the function is expected to fetch an integer.
in	**ValueMaxSize**	Maximum number of characters used for the parameter value in the configuration file.

Returns

Error code or success (0)

Result ConfigFile_GetSectionNamesFromParameter(const char * Filename, const char * Parameter, char ** Section, uint32_t * u32NbSections);

Gets the names of sections having parameter Parameter declared in them. The function stores the names in variable pointed to by Section and the number of sections found in variable pointed to by u32NbSections.

Parameters

In	**Filename**	Name of file to parse
In	**Parameter**	String holding the parameter name. The exact same string is expected to be found in the configuration file.
Out	**Section**	Pointer to strings that will hold the section names.
in	**u32NbSection**	Number of sections in which parameter *Parameter* was found.

Returns

Error code or success (0)