Quick Start Guide

Ryan Putz

August 18, 2011

Contents

1 Introduction

The purpose of this document is to train users in the proper useage of the Fermilab Hierarchical Configuration Language (FHiCL). This Quick Start Guide will explain and demonstrate the major syntax and semantics of the language so that users may create valid FHiCL documents with ease.

2 Comments

FHiCL comment notation is a combination of PHP and C/C++ comment syntax. FHiCL comments have two notations:

- The pound/hash sign (#)
- Two foreslashes (//)

The former notation is the same notation used in such programming and scripting languages as Perl, Python, and Ruby.
The latter notation is the C++ inline comment notation.

3 Associations

3.1 Names

FHiCL names are simple, non-quoted strings. Names may not begin with a digit, but may begin with an underscore. FHiCL names may not contain whitespace or escaped characters
Illustrated in Example 1.1.0, you will find a list of examples of valid FHiCL name values.
Example 1.1.0:

```
a
name
the_dog
_a_cat
a_fish_
_a_bird_
```

There is also an advanced type of FHiCL name called *hierarchical name* or *hname*, for short. These hnames are compound names linked together through the use of a *dot index* or a *bracket index*. Hnames are used to access members of FHiCL containers and their associated values.
Example 1.1.1:

```
name.first = "beth"
name.last = "johnson"
scores[0] = 5
scores[3] = 6
```

We will see more in-depth examples of hname useage later on.

3.2 Values

In contrast, FHiCL values are extremely diverse. FHiCL values can be any one of the following:

- Single character (quoted or unquoted)
- String (quoted or unquoted)
- Number
- Table
- Sequence
- Document

Single-quoted strings are taken verbatim and may contain escaped or printable characters as well as whitespace.
Double-quoted strings may contain escaped and printable characters, but are not quoted verbatim. Escaped characters are processed in the string. Double-quoted strings may contain whitespace.
In Example 1.2.0, there are a few examples of valid FHiCL values. Example 1.2.0:

```
a
ab
string
"string"
'string'
'string\n'
12345
"123abc"
'123abc'
```

3.3 Associations

In most programming languages there is the idea of an *assignment*. In FHiCL, however, instead of assigning a value to a name, we are *associating* a value with a name; much like the relationship between a key and a value in a C++ std::map or a Python dictionary.
The basic association is the backbone of the FHiCL syntax. A basic FHiCL association has three parts:

1. A name followed by...
2. The colon symbol (:); followed by...

3. A value.

Or in EBNF it would look like this:

```
association => name COLON value
```

There are slight variations on this model used throughout the language, but the basic premise does not change. We will see examples of these variations later on.

3.4 Examples

Let's say that we want to declare three new FHiCL elements: a, b, c with values of 1, 2, 3, respectively. Here is how that would look in a valid FHiCL document:
Example 1.4.0:

```
a : 1
b : 2
c : 3
```

Now we have three new FHiCL name-value pairs! Please note that in *most* aspects of FHiCL, the use of whitespace is optional.
So this would also be a valid way to write the associations in Example 1.0:
Example 1.4.1:

```
a:1
b:2
c:3
```

Here are a few more examples of valid FHiCL associations using various types of strings:
Example 1.4.2:

```
dog : "Wish Bone"
cat : 'QT McWhiskers'
fish : Klaus
```

Please note that unquoted string values must be *simple* in that they may only contain the characters A-Z, a-z, or underscores (_). Double-quoted strings may contain special escaped characters, and single-quoted strings are quoted verbatim.

4 Overrides

In FHiCL, the term *override* has special meaning and its use is two-fold:

1. Reassociation of an existing name to a new value.
2. Creation of a new element within a table or sequence.

The syntax for a FHiCL override statement is identical to that of a FHiCL association. The only difference between the two is that in an override statement, the *name* portion of the statement must be the name of a pre-existing element which exists at the same level of scope as the override statement.
Please note that the use of an hname allows an override statement to descend through layers of scope, but there is no way to ascend through layers of scope.
Example 2.0.0:

```
x : 5
x : 6
```

Example 2.0.1

```
tab1: { a:1 b:2 }
tab1.c : 3
```

5 Tables And Sequences

Two major constructs in FHiCL are the *table* and the *sequence*. Each of these containters has a unique syntax for declaration and access.

5.1 Tables

A FHiCL table is a container for name-value pairs and is denoted by (possibly empty) braces.
Example 3.1.0

```
{
   a : 5
   b : 6
   c : 7
}
```

Elements of a FHiCL table may *not* be comma-separated. The elements may be separated by either whitespace or newline. Therefore it would also be valid to write *table1* like this:
Example 3.1.1

```
{ a : 5 b : 6 c : 7 }
OR
{ a:5 b:6 c:7 }
OR
{
   a:5
   b:6
   c:7
}
NOT
{ a:5, b:6, c:7}
```

FHiCL tables also have a unique quality in that they may contain comments. This is true only if the elements of the table are line-separated. Example 3.1.2

```
{
   a : 5
   #This is a comment
   b : 6
   //This is a comment
}

NOT

{ a:5 #this is a comment b:6}
```

The second case is invalid since FHiCL comments take the rest of the line, the closing brace for the table will be interpreted as part of the comment.
In order to acces members of a FHiCL table, use of an *hname* with the *dot index* is needed.
Useage of the *dot index* notation requires that the table be associated with a name.
For instance, if we wanted to access *a* from *table1* in Example 3.1.0, and override the value of *a* to 4, it would look like this:
Example 3.1.3:

```
table1:{
                            a : 5
                            b : 6
                            c : 7
                          }
table1.a : 4
```

Similarly, if we wanted to add or *override* in a fourth member to *table1* (we'll call it *d* and we'll give it a value of 8) it would look like this:
Example 3.1.4:

```
table1.d : 8
```

5.2 Sequences

A FHiCL sequence is a container of values and is denoted by (possibly empty) brackets.
Example 3.2.0:

```
[ 1, 2, 3, 4 ]

OR

[
```

```
   1,
   2,
   3,
   4
]
```

Elements of a sequence *must* be separated by commas. Each element in a sequence is a value, and so, it is possible to have a sequence of tables such as:

```
seq:[ { a:1 b:2 }, { c:3 d:4 } ]
```

Also, please note that FHiCL sequences may not contain name-value pairs.
FHiCL sequences may not contain comments.
In order to access members of a FHiCL sequence, use of an *hname* with the *bracket index* is needed.
Useage of the *bracket index* requires that the sequence be associated with a name.
For example, if we wanted to access the first element in seq1 from Example 3.2.0, and override the value to 5, it would look like this:
Example 3.2.1:

```
seq1[0] : 5
```

Similarly if we wanted to add a fifth value (let's say, the number 6?) to seq1, here's how it would look:
Example 3.2.2:

```
seq1[4] : 6
```

Please note that indices of FHiCL sequences begin at zero, not one.

6 Include Statements

In order to import values from one FHiCL document into another, use of the FHiCL include statement is necessary. The behavior of this include statement is similar, and yet still different from the C/C++ include statement from which it was borrowed.
The syntax for a FHiCL include statement begins with a pound or hash symbol (#), followed by the unquoted string "include". The include is followed by *exactly one white space* and then a quoted string containing the full name of the target FHiCL document.
Example 4.1.0:

```
#include "filename.fcl"
```

Since the include statement shares a common starting character with that of one of the FHiCL comment notations, if the syntax of the include statement is incorrect, a FHiCL parser will treat the faulty include statement as a comment. If the file name within the quotes is invalid, however, the include statement will *not* be treated as a comment.

7 Prologue

FHiCL documents may contain prologue sections which may contain declarations used to create a standard configuration for a FHiCL document.
The beginning of a prologue section is denoted by: *BEGINPROLOGUE* and the end is denoted by: *ENDPROLOGUE*.
Example 5.1.0:

```
BEGINPROLOGUE
   x : 5
   y : 6
ENDPROLOGUE
```

All elements declared between the beginning and the end do not exist within the parameter set unless specifically referenced within the document.
Elements within a prologue section may be referenced, but may not be modified in any way otherwise. For instance, take the prologue section from Example 5.1.0.
We can reference both x and y in the prologue section by using the reference notation:
Example 5.1.1:

```
z : @local::x
u : @local::y
```

However we cannot modify x or y in any way. Access to values within a prolog works exactly like it does when the referenced value is in the document. You use the reference notation *@local::* folowed by the target value you wish to reference. It is important to note that you *MAY NOT* override values within a prolog from outside of a prolog. The only way to override prolog values is to use a reference within either the same prolog, or another prolog. Statements that appear to be attempting to override prolog values are treated as new associations, therefore creating new associations in the parameter set.

8 References

If at any point it becomes necessary to associate a name with a value of an existing name-value pair, or to assign an existing value as a member of a sequence, the use of the FHiCL *Reference* notation is needed.
There are two versions of the reference notation in FHiCL, one being a local reference, the other being a call or fetch to a predetermined database:

- @local::
- @db::

References may appear wherever a value normally is expected. The only prerequesite being that the name used within the reference notation must be the

name of an existing element either within the current FHiCL document or within the called database.
Example 6.1.0:

```
x : 5
y : @local::x
z : @db::t
```

In this example, the value of y is a reference to the pre-existing name-value pair $x : 5$. The element named z's value is a reference to a name-value pair with name t which is to be found in a database.
The names used within a FHiCL reference notation must be extremely specific in order to reference the correct element.
Example 6.1.1:

```
x : 5
x : 6
tab1 :
{
   x : 1
   y : 2
   z : 3
}
y : @local::x
```

In Example 6.1.1, we start with a definition $x : 5$, and then we override the value of x with the number 6. The value of y in this case will be a reference to the most recently encountered instance of x within the same scope as y.
Essentially this means that the reference to x here will have the value of $x : 6$.

9 Documents

FHiCL documents closely resemble FHiCL tables in that they are, for the most part, containers of name-value pairs. Unlike tables, however, FHiCL documents do not have braces around their elements and documents may contain prologue sections.
A valid FHiCL document might look something like this:
Example 7.1.0:

```
BEGINPROLOGUE
   x : 5
   y : 6
ENDPROLOGUE

tab1 :
{
   a : 1
```

```
   b : 2
   c : 3
    }
z : @local::x
y : @local::y
```

10 Conclussion

This concludes the FHiCL Quick Start document. The definitions and examples given above will allow users to create their own FHiCL documents with ease, and should provide a quick reference for users who are familiar with the language but are in need of a primer.