Date:	2019-05-07
Author:	Yann Weyland, IPG Solution Engineering
Release No.:	CarMaker 6.x – 8.x

How to handle InfoFiles from Matlab environment?

CarMaker bases its parameterizing system on the InfoFile. The chapter "InfoFile Module" in the Programmer's Guide describes the format of the InfoFile and the API for C- or Tcl-code level. But there is also an API available for Matlab.

It is necessary to run the script "cmenv.m" in the source folder of a CarMaker project directory to get access to all the commands of InfoFile-API in Matlab. All manipulations of InfoFile happen at the memory level, which means the content of an InfoFile needs to be copied to memory first. After finishing, a memory dump creates the corresponding file.
Please find below a common example to implement this approach.

```
1:  % create a new infofile handle
2:  ifid = ifile new();
3:
4:  % load file content into memory
5:  InfNameIn = 'MyInfInput';
6:  NOTOK = ifile read(ifid, InfNameIn);
7:  if NOTOK
8:    error(sprintf('problem loading %s!', InfNameIn));
9:  end
10:
11: % save memory data to file system
12: InfNameOut = 'MyInfOutput';
13: NOTOK = ifile write (ifid, InfNameOut);
14: if NOTOK
15:   error(sprintf('problem saving %s!', InfNameOut));
16: end
17:
18: % delete this handle (when finished)
19: ifile_delete(ifid);
```

Example 1 – Matlab code for basic handling of InfoFiles

The generation of a handler is the first task with the command "ifile_new" (line 2). Similarly, releasing a handler no longer needed is performed at the end with the command "ifile_delete" (line 19). Between these two tasks, it is possible to handle the data related to the InfoFile. This can be, for example,

- to import the content of a file on the file system into the memory pointed by the handler with the command "ifile_read" (line 6)
- or to export the memory data to a file on the file system with the command "ifile_write" (line 13).

Such commands mostly have a return parameter telling whether the command has run successfully or not. Therefore, consider a proper implementation to check the corresponding parameter.

Write and read InfoFile keys

The most basic and important functionality is to get or set InfoFile keys (in memory).
Setting a key automatically creates the key in memory, when not already existing, or modifies the existing one.

The next example shows how to read and write line keys with the commands “ifile_getstr” (line 9) and “ifile_setstr” (line 5). The “set” command accepts as class for the value argument string, integer and floating values.

```
1:  % create and check some dummy line keys with dummy values
2:  keys = {'dummy' 'constants' 'x-axis' 'random' 'answer'};
3:  values = {'foo bar' 'pi 3.14159 e 2.71828' '1 0 0' rand() 42};
4:  for i=1:5
5:    NOTOK = ifile setstr(ifid, keys{i}, values{i});
6:    if NOTOK
7:      error(sprintf('problem writing key %s!', keys{i}));
8:    end
9:    [VALUE NOTOK] = ifile getstr(ifid, keys{i});
10:   if NOTOK
11:     error(sprintf('problem reading key %s!', keys{i}));
12:   end
13: end
```

Example 2 – Matlab code for writing and reading line keys

For more complex data, the InfoFile format supports multiline text keys and their corresponding commands work very closely to above example with line keys.
The next example shows how to read and write text keys with the commands “ifile_gettxt” (line 9) and “ifile_settxt” (line 5). Please call the commands “ifile_gettxt” and “ifile_settxt” to read and write text keys instead of the commands mentioned in the previous example.

```
1:  % create and check some text keys
2:  txtkeys = {'key.name' 'key.values'};
3:  txtvalues = {keys values};
4:  for i=1:2
5:    NOTOK = ifile settxt(ifid, txtkeys{i}, txtvalues{i});
6:    if NOTOK
7:      error(sprintf('problem writing text key %s!', txtkeys{i}));
8:    end
9:    [VALUE NOTOK] = ifile gettxt(ifid, txtkeys{i});
10:   if NOTOK
11:     error(sprintf('problem reading text key %s!', txtkeys{i}));
12:   end
13: end
```

Example 3 – Matlab code for writing and reading text keys

Working with Matlab mostly leads to handle matrices. The InfoFile interface for Matlab especially provides two additional commands for this purpose.
The next example shows how to read and write matrix keys with the commands “ifile_getmat” (line 9) and “ifile_setmat” (line 5). Please call the commands “ifile_getmat” and “ifile_setmat” to read and write matrix keys instead of the commands mentioned in the previous example.

```
1:  % create and check some matrix keys
2:  matkeys = {'matrix3x3' 'matrix3D'};
3:  matvalues = {rand(3,3) rand(2,4,3)};
4:  for i=1:2
5:    NOTOK = ifile setmat(ifid, matkeys{i}, matvalues{i});
6:    if NOTOK
7:      error(sprintf('problem writing matrix key %s!', matkeys{i}));
8:    end
9:    [VALUE NOTOK] = ifile getmat(ifid, matkeys{i}, double(0), 1);
10:   if NOTOK
11:     error(sprintf('problem reading matrix key %s!', matkeys{i}));
12:   end
13: end
```

Example 4 – Matlab code for writing and reading matrix keys

Practically a matrix may be a vector (row or column) or a simple 2D matrix or else even a multidimensional matrix but the benefit of the matrix commands “ifile_getmat” and “ifile_setmat” are their usage remains the same for all dimensions of the matrix.

The complexity is handled by adding the header line describing the structure of the matrix before other lines corresponding to the matrix coefficients list is represented.

```
1:  matrix3D:
2:          #MATRIX1 real double 24 3 2 4 3
3:          0.76551679 0.7951999 0.1868726 0.4897644 0.4455862 0.64631301
4:          0.70936483 0.75468668 0.27602508 0.67970268 0.655098 0.16261174
5:          0.11899768 0.49836405 0.95974396 0.34038573 0.58526775 0.22381194
6:          0.75126706 0.75468668 0.50595705 0.69907672 0.89090325 0.95929143
```

Example 5 – matrix key stored in its InfoFile as set by its Matlab command

This header (line 2) includes at its end a list of (at least 3) values.

- The first value indicates the number of matrix coefficients (24 coefficients).
- The second value indicates the number of matrix dimensions (3 dimensions).
- The last values indicate the number of matrix coefficient sets for each matrix dimension, starting from first dimension to last one (2 for the 1st dimension, 4 for the 2nd and 3 for the 3rd: $\underbrace{2 \times 4 \times 3}_{3} = 24$).

As long as this header line is a part of the matrix key and the coefficients list complies with it, no fallback information is necessary from the user by reading the matrix key. Therefore, the two last arguments can be kept as set in the 4th example (line 9).