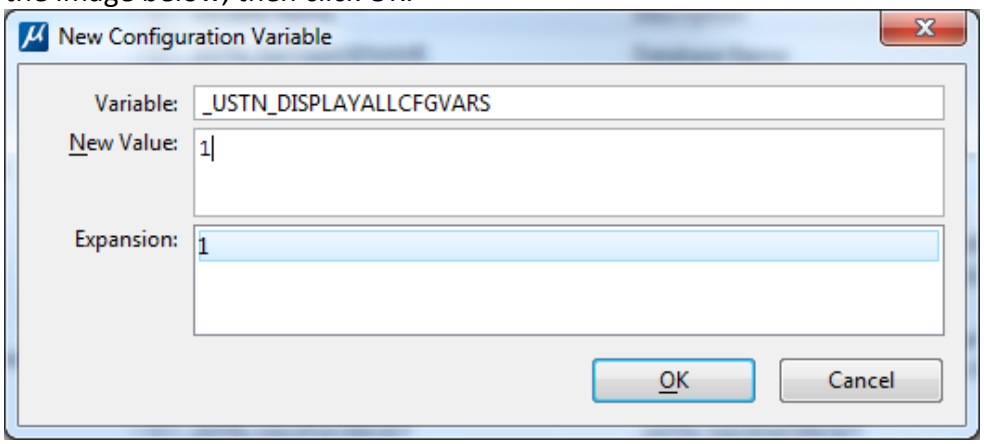
CONFIGURATION FOR BENTLEY PRODUCTS

By John Davidson

Learning how to manage Bentley Products Configuration can be a daunting task when first introduced to it. Thoughts go through your mind like, how am I going to learn to manage this, it looks like computer programming and I'm not a programmer. Well, you are sort of right, but this configuration system is extremely easy "programming", in plain English with a very few exceptions.

We will now make the first step towards being a configuration wizard. Simply follow these steps and you will soon be one.

1. Learn how to make sense of the Configuration dialog.
    a. The dialog shows us the current settings for most of the configuration variables. What are variables did you say? A variable is a storage location with a symbolic name, which contains some information referred to as a value. We use variables to hold values that may change depending on varying circumstances, like, which option you picked in the Workspace or Workset dropdown lists, or what is set in the configuration files, etc. These variables tell the product where to find things, which options you have chosen, and all the necessary information to meet your configured requirements.
    b. The Configuration dialog is used to edit, define, save, and delete Configuration Variables. There is very good help for this included with all the products, search the help for "configuration variables dialog". The dialog may be slightly different in some products so best to use your products help for this.
    c. Changes done through the dialog get saved to the users private configuration file and are not seen by any other user, if you require to make changes available to other users, the changes need to be save into the product configuration files, in the company configuration.
2. In order to make changes to configuration, you need to know which files to edit and of course how to make the product understand what you want it to do. That means you will need to understand the configuration flow when the product starts and if changes are made during the product instance.
    a. When a product starts, it always runs the main configuration file that controls the whole process and it calls all the other configuration files in the correct order. The main configuration file is called msconfig.cfg and resides in the _ROOTDIR/config/ folder. To find what the actual folder name is in the Configuration dialog, you need to set a configuration variable so that it is visible, so here is your first exercise. Click the New button and enter the information shown in the image below, then click OK.



    You should then see lots of variables starting with "_USTN_", click on _ROOTDIR and in the expansion window you will see the folder name.
    b. When you find the msconfig.cfg file, remember never, I repeat NEVER, edit this file, it is exactly the same contents, for any given version set of Bentley products, that includes MicroStation, like OpenRoads, OpenRail, OpenBuildings, etc.

c. Open it up with NotePad or similar application, it is a plain text file, read it through to the end. Try to understand what it is directing the product to do, but you will probably need to read the next section on **learning the language** to get a full understanding.

3. **Learn the language**: What does each keyword or operative mean, to understand this you will need to learn that the text in these files needs to follow a very strict syntax for the product to understand what you want. If there are any errors or the product cannot decipher it, you will get a CMD window showing what may have gone wrong, or the line number in the config file that is not correct.
   So here is your next project, read the following pages until you understand it in general, you will make mistakes and probably have to reread them again several times before you can consider yourself confident.

The next few pages are straight out of the product help file, and explain the language fundamentals.

Configuration File Syntax

Configuration Files consist of statements of following types:

- Flow directives that control the flow through Configuration File
- Variable directives that control certain aspects of Configuration Variables
- Assignment statements that set the value of Configuration Variables
- Expressions and operators that manipulate strings or Configuration Variables to yield results that can be used in directives or assignments

When files or directories are specified in Configuration Files, forward slashes are used as directory separators, and whenever a directory is specified as the value of a Configuration File, it is followed by a trailing forward slash. For example, the following statement sets the Configuration Variable MS_DEF to the c:\users\John.Smith\Documents directory:

MS_DEF = c:/users/John.Smith/Documents/

Tip: Leaving off the trailing slash is a common error.

Configuration Variables are often defined in terms of other Configuration Variables. Following are the syntaxes for defining a Configuration Variable in terms of another Configuration Variable:

Syntax	Meaning
`$(<CfgVarName>)`	The plain parentheses cause the expression to be stored as the Configuration Variable definition verbatim and evaluated only when the value of the Configuration Variable is eventually needed during program execution. This allows Configuration Variable definitions to use other Configuration Variables that have not yet been defined. This is the most commonly used syntax.
`${<CfgVarName>}`	The curly braces cause the expression to be immediately evaluated, and the result stored as the Configuration Variable definition. Therefore, any Configuration variable contained within the curly braces must have been defined by an earlier assignment.

Flow Directives and Variable Directives

Flow directives: control the way that the product processes configuration files. Flow directives always begin with `%`. The following flow directives are available:

Flow directive	Syntax	Meaning
`%include`	`%include<filespec>`	Includes one or more configuration files before proceeding with the next line in this configuration file. `<filespec>` can specify a single file, or can include wildcard characters to include many files. `<filespec>` can contain a Configuration Variable. For example: `%include $(_USTN_WORKSPACECFG)`
	`%include<filespec> level <levelspec>`	The `%include` directive can include a Configuration Variable level at which the assignments in the included file are applied (until another level directive is encountered). For example: `%include $(_USTN_ROLECFG) level Role`
`%if`	`%if <expression>`	If `<expression>` evaluates to true, continue on the next line, otherwise skip to the matching `%else`, `%elif`, or `%endif` statement. See Operators for the syntax of `<expression>`. For example: `%if $(PHASE)=="Final"`
`%ifdef`	`%ifdef<cfgvar>`	If `<cfgvar>` is defined, continue on the next line, otherwise skip to the matching `%else`, `%elif`, or `%endif` statement. For example: `%ifdef _USTN_ROLECFG`
`%ifndef`	`%ifndef<cfgvar>`	If `<cfgvar>` is not defined, continue on the next line, otherwise skip to the matching `%else`, `%elif`, or `%endif` statement. For example: `%ifndef MS_DEF`
`%else`	`%else`	An `%if`, `%ifdef`, or `%ifndef` statement that evaluates to false continues at the line following a matched `%else` statement, if there is one.
`%elif`	`%elif <expression>`	An `%if`, `%ifdef`, or `%ifndef` statement that evaluates to false continues by evaluating `<expression>` at the first matched `%elif` statement, and then either continues processing at the line following if expression evaluates to true, or skips to the next `%elif`, `%else`, or `%endif` statement. For example: `%elif defined (MS_RFDIR)`
`%endif`	`%endif`	The statement that indicates the end of the conditional block for an `%if`, `%ifdef`, or `%ifndef` statement.
`%echo`	`%echo<message>`	Shows the contents of `<message>` in MicroStation's text window and continues processing. For example: `%echo $(MS_DEF)`
`%error`	`%error<message>`	Causes processing to stop, reporting the contents of `<message>` as the error. For example: `%error unexpected value`

Variable Directives:

The following variable directives are supported:

Variable directive	Syntax	Definition
`%lock`	`%lock<cfgvar>`	Locks the Configuration Variable `<cfgvar>` so that it cannot be changed.
`%undef`	`%undef<cfgvar>`	Discards the value of the Configuration Variable and sets it to undefined.
`%level`	`%level<newLevel>`	Specifies the level at which any following Configuration Variable definitions are to be applied. The `<newlevel>` argument should be one of the following: • System • Application • Organization • WorkSpace • WorkSet • Role • User Note: The above arguments are *not* case-sensitive.

Assignment Statements

The following table shows the different ways that a Configuration Variable can be defined.

Assignment operator	Definition
=	Assign the Configuration Variable at the current level, regardless of whether it is currently defined. For example: `MS_SHEETMODELNAME = 2D Sheet`
:	Assign the Configuration Variable at the current level, but only if it is not already defined. For example: `MS_BACKUP : $(_USTN_OUT)`
>	Appends the operand to the existing definition of the variable, separating the existing value and the operand by semicolons (treating the variable as a path). For example: `MS_RFDIR > $(_USTN_WORKSETROOT)Borders/`
<	Prepends the operand to the existing definition of the variable, separating the existing value and the operand by semicolons (treating the variable as a path). For example: `MS_RFDIR < $(_USTN_WORKSETROOT)Borders/`
+	Appends the operand to the existing definition of the variable without separator. For example: `_USTN_WORKSETDESCR + In Development`

Operators

The following table shows operators that can be used in Configuration Variable definitions and how they are interpreted as a Configuration Variable is expanded. In the examples below, assume that `$(USTN_WORKSETCFG)` is defined as `g:/Clients/DeptOfTransportation/WorkSets/Highway131.cfg`

Operator	Syntax	Definition
basename	basename(<expression>)	Returns the filename of <expression> without directory or extension. For example: `WORKSETNAME = basename ($(_USTN_WORKSETCFG))` gives `Highway131`
concat	concat(<arg1>,<arg2>...)	Returns the concatenation of the arguments, similar to the + operator, but allows multiple arguments. For example: `LIST = concat (CFG1,CFG2,CFG3)`
devdir	devdir(<expression>)	Returns the device and directory of <expression>, including a trailing directory separator. For example: `WORKSETDIR = devdir ($(_USTN_WORKSETCFG))` gives `g:\Clients\DeptOfTransportation\WorkSets\`
dev	dev(<expression>)	Returns the device (for example, c:) of <expression>. For example: `WORKSETDEV = dev ($(_USTN_WORKSETCFG))` gives `g:`
dir	dir(<expression>)	Returns the directory (without the device) of <expression>. For example: `WORKSETDIR = dir ($(_USTN_WORKSETCFG))` gives `\Clients\DeptOfTransportation\WorkSets\`
ext	ext(<expression>)	Returns the file extension of <expression>. For example: `WORKSETEXT = ext ($(_USTN_WORKSETCFG))` gives `.cfg`
filename	filename(<expression>)	Returns the filename and extension of <expression>. For example: `WORKSETFILE = filename ($(_USTN_WORKSETCFG))` gives `Highway131.cfg`
first	first(<expression>)	Returns the first portion of an expression (that is, the part preceding the first semicolon). For example: `FIRSTREFDIR = first ($(MS_RFDIR))`
firstdirpiece	firstdirpiece(<expression>)	Returns the root directory (without device) of <expression>. For example: `WORKSETROOT = firstdirpiece ($(_USTN_WORKSETCFG))` gives `Clients`
lastdirpiece	lastdirpiece(<expression>)	Returns the portion of the directory closest to the file in <expression>. For example: `WORKSETPAR = lastdirpiece ($(_USTN_WORKSETCFG))` gives `WorkSets`

Operator	Syntax	Definition
noext	noext(<expression>)	Returns the full path of <expression>, omitting the extension. WORKSETFILEROOT = noext ($(_USTN_WORKSETCFG)) gives g:\Clients\DeptOfTransportation\WorkSets\Highway131
parentdevdir	parentdevdir(<expression>)	Returns the parent directory, including the device, of <expression>. For example: WORKSETPDD = parentdevdir ($(_USTN_WORKSETCFG)) gives g:\Clients\DeptOfTransportation\
parentdir	parentdir(<expression>)	Returns the parent directory, excluding the device, of <expression> WORKSETPD = parentdir ($(_USTN_WORKSETCFG)) gives \Clients\DeptOfTransportation\
registryread	registryread(regvar)	Returns the contents of the registry variable regvar. For example: PWDIR=registryread("HKEY_CURRENT_USER\SOFTWARE\Bentley\ProjectWise\Path")

Logical Expressions

Logical expressions are compound relational expressions formed by combining simple relational expressions with logical operators. The logical_and operator (AND, .AND. or &&) and the logical_or operator (OR, .OR. or ||) are both binary operators. Both operands must be simple relational expressions or logical expressions that evaluate to true or false. The logical_not operator (.NOT.) requires only a single operand. The following list contains descriptions of each of the logical operators and some example logical expressions.

Operator	Result
&&	Evaluates to true if and only if both operands are true. Otherwise it evaluates to false.
||	Evaluates to true if either operand is true. Evaluates to false if and only if both operands are false.
!	Evaluates to true if the single operand is false. Evaluates to false if the operand is true.

Examples:

((type == line) || (type == line_string))
((level == 15) && (type i text))
(!(color == 5))

4. *msconfig.cfg file:*

Read the msconfig.cfg file and learn what each instruction does, some lines are just setting variables, or checking what information variables contain, and some are redirecting the configuration flow to other configuration files, to run them and then return to the next line in msconfig.cfg.

a. Read the first 80 odd lines which explain some more of the syntax and meaning of special characters and brackets etc. and in particular the level variable information. The configuration is divided into 7 levels System (0) Application (1) Organization (2) Workspace (3) Workset (4) Role (5) and User (6), and the level variable sets the configuration level for the processes following it.
b. The # character indicates that everything after it on the current line, is ignored by the process, and is used for comments to let the reader know what the process is doing, or simply to make the code easier to read.
c. Processing should be done in configuration level, or the level variable set to the appropriate level for the following processes.
d. The following lines are typical of how to test if something exists and if it does include it, or run the process/s between the test line and %endif. For each %if there must be a corresponding %endif or an error will occur.

```
%if exists ($(_USTN_INSTALLED_CONFIGURATION)ConfigurationSetup.cfg) # Test if exists
   %  include $(_USTN_INSTALLED_CONFIGURATION)ConfigurationSetup.cfg # If true, run this process
%endif
```

e. The %include tells the processor to run the nominated cfg file, and then return flow back to the next line.
f. If you are using Connect Edition, see if the flow diagram of the configuration process for Connect Edition products on the next page matches what you have found.

Once you can read and understand the configuration file syntax, you should be able to understand how all the configuration files work toward the final configuration.

The next part of the process is to learn where to find and understand the uses of all the available configuration variables that are built into each application. Search for **Configuration Variables** in the product help**.** They are not all in one place so you need to look at many pages to find them all, but they are in logically named groups.

In addition to these you can create your own variables and use them to achieve your desired result.

You are not limited to using just the supplied configuration files, you can also create your own, and either put them in the logical folders where all *.cfg files are processed, or you can run them from existing .cfg files like the example above.

Always use variables for addressing locations and anything else that is already stored in a variable so that if anything changes, it will change throughout the configuration. The only hard coded information should be the initial variable loading.

msconfig.cfg Is exactly the same for all Bentley CE products
of the matching MSCONFIG version

By John Davidson (BUGVA)