

Building a Corporate MicroStation Workspace

Prepared for:

2007 MC²
Mid Continent
MicroStation Community
August 16, 2007
Overland Park, KS

Speaker: Mark Mates, ProSoft

Utilizing the basic components of a workspace and specific configuration variables, you can build an efficient corporate MicroStation® workspace to provide users with the access to all necessary corporate standard and project data resources. The workspace should be easy to update and maintain. This session is Part 2 of a two-part series and is based on excerpts from the **MicroStation V8 CAD Manager** manual by ProSoft.



ProSoft

1776 North State, Suite 200
Orem, Utah 84057
(888) 263-0393

www.prosoftnet.com
info@prosoftnet.com

MicroStation is a registered trademark of Bentley Systems, Inc.

Introduction to Corporate Workspaces

Building a corporate workspace requires a variety of data resources, an understanding of the basic workspace components, and knowledge of configuration variables. Then all of the pieces can be put together into a corporate workspace that provides users with access to all necessary corporate standard and project data resources. The trick, however, is to design the workspace so that it will be easy to update and maintain. This session examines some of the tools that make this possible. Some of the key issues include:

- Understanding configuration variables
- Setting up a site configuration
- Segregating customization from the core program
- Creating and managing projects
- Setting up users
- Implementing a corporate standard interface

Understanding Configuration Variables

MicroStation uses configuration variables to set such things as default search paths, output directories, MDL applications to load, user interfaces to display and much more. Skillful manipulation of configuration variables can result in a virtually unlimited number of useful, highly-customized configurations of MicroStation.

Why Use Configuration Variables?

Configuration variables offer tremendous flexibility to site administrators by limiting the use of hard-coded paths. The relative path defined in one configuration variable can be based on the definition of another variable, which in turn may be based on another configuration variable. This layering of relative paths means that a change to one variable will automatically redirect the other variables that are based on the modified variable.



If a configuration variable is not set at any of these levels, MicroStation will look for an operating system variable with the same name.

Configuration Variable Hierarchy

As MicroStation processes configuration variables, it prioritizes them according to a predefined hierarchy consisting of five main “levels”: **system**, **application**, **site**, **project** and **user**. A variable set at a lower level can be overridden by the same variable set a higher level.

System

System-level variables are processed first and have the lowest priority. Generally, these variables are set by MicroStation and should not be modified directly.

Application

Application-level variables are processed second and are assigned higher priority than system-level variables. These variables are set by installed applications that run inside of MicroStation, such as TriForma, InRoads, GEOPAK, etc.

Site

Site-level variables are processed third and are assigned a higher priority than system- and application-level variables. These variables are set by a site administrator to establish a corporate standard configuration of MicroStation.

Project

Project-level variables are actually processed last, but are next in terms of priority. These variables are set by a site administrator or project manager to customize the behavior of MicroStation for a specific project.

User

User-level variables are processed fourth (between **site** and **project**), but have the highest priority. These variables are set by the user. However, a site administrator can set and lock a variable to prevent its modification by the user.

The figure below shows the processing order and priority of each level in the variable hierarchy graphically.

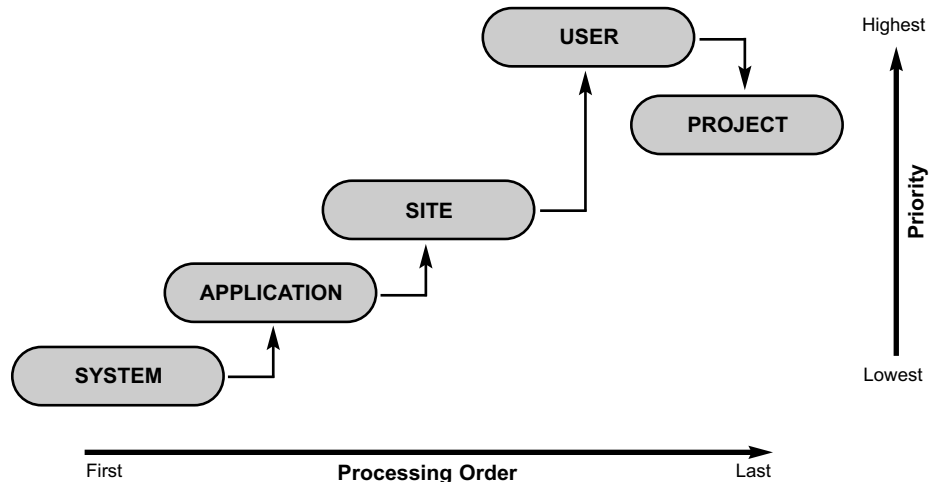


Figure 1
Configuration Variable Hierarchy



You can bypass **mslocal.cfg** by setting up a separate startup configuration file.

Configuration Files

Configuration variables are stored in text files called “configuration files,” most of which have a .CFG file extension. The variables contained in a configuration file will be processed at one of the levels described previously, depending on the directory in which the configuration file is stored. Following is a brief description of three important configuration files.

MSLOCAL.CFG

Mslocal.cfg is, by default, the first configuration file read by MicroStation. This file is located in the **\Program Files\Bentley\Program\MicroStation\config** directory. An unmodified **mslocal.cfg** file will include the three lines shown below (variable definitions may be different, depending on where the software is installed):

```
MSDIR=C:/Program Files/Bentley/program/microstation/  
_USTN_WORKSPACEROOT:C:/Program Files/Bentley/Workspace/  
%include $(MSDIR)/CONFIG/msconfig.cfg
```

The first line sets the root installation directory for MicroStation. The second line sets a preliminary location for the main workspace directory. The last line points to the location of the next configuration file to be processed, **msconfig.cfg**.

MSCONFIG.CFG

Msconfig.cfg is the main MicroStation configuration file. By default, this file is also located in the **\Program Files\Bentley\Program\MicroStation\config** directory. This file has three main purposes: 1) it sets some default variable definitions, 2) it specifies which configuration files should be included during variable processing and 3) it sets the level at which the variables contained in these configuration files will be processed. This file should not be modified. Configuration changes should be made in one of the configuration files included by **msconfig.cfg**.

STANDARDS.CFG

Standards.cfg contains site-level variables that are set for all users at the site. Site administrators can use this file to point to the location where corporate standard data resources are stored. By default, **standards.cfg** is located in the **\Program Files\Bentley\Workspace\standards** directory.

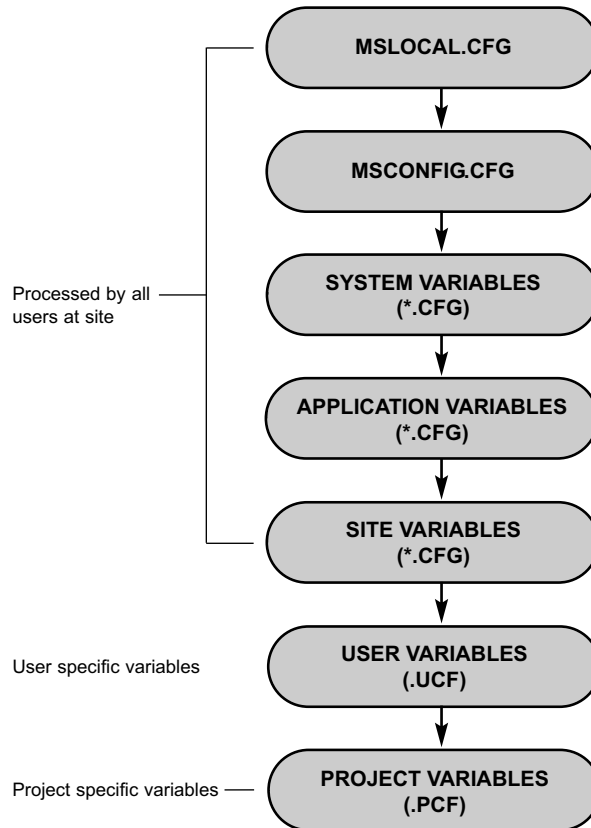


When MicroStation encounters more than one configuration file in the same directory, it processes them in alphabetical order. This means that a configuration file beginning with the letter “A” would be processed before a file beginning with the letter “B.” If a conflicting variable definition exists, the last configuration file processed takes precedence.

Variable Processing At Startup

When MicroStation is launched, the program educates itself by processing all included configuration files in a specific order beginning with the **mslocal.cfg** configuration file. **Mslocal.cfg** points to **msconfig.cfg**. **Msconfig.cfg** tells MicroStation which additional configuration files to process. MicroStation then processes system configuration files first, followed by application, site, user and project configuration files (in that order). The figure below illustrates this process.

Figure 2
Variable Processing at Startup





You can generate a complete list of MicroStation variables by entering the following in the Key-in window: **mdl load cfgvars printCfgVarResource**. This will generate two text files in the **\Program Files\Bentley\Program\MicroStation** directory: **cfgvars.txt** and **cfgvlong.txt**. **Cfgvars.txt** lists each variable, along with its variable type and a short description. It also specifies whether changing the variable requires a restart. **Cfgvlong.txt** lists each variable with a longer description.



Filename/Filelist variables support the use of wildcard characters in their definitions.

Types of Configuration Variables

There are five basic types of configuration variables: **path**, **directory**, **filename/filelist**, **keyword** and **boolean**. The following is a brief definition of each variable type, along with some examples.

Path Variables

Path variables specify which directory or directories MicroStation should search in to find a particular file or resource. For example, **MS_RFDIR** tells MicroStation where to look for a reference file. If the reference file exists in any of the specified directories, MicroStation will find and attach the file.

Directory Variables

Directory variables specify where files should be saved. Unlike path variables allowing multiple paths, directory variables indicate one directory. **MS_PLTFILES** is a good example. When plotting to a file, the directory specified in the **MS_PLTFILES** variable is automatically selected in the Save Print As dialog box.

Filename/Filelist Variables

Filename and filelist variables specify which file(s) to use for a particular purpose. For example, **MS_ACCUDRAWKEYS** specifies the text file to use for AccuDraw shortcuts..

Keyword Variables

Keyword variables specify default settings. For example, **MS_WORKMODE** specifies the active workmode. Valid keywords are “DGN”, “V7”, and “DWG”.

Boolean Variables

Boolean variables toggle settings on and off. Generally, if the variable is undefined or is set to **0**, it is considered “off.” If it is set to **1**, it is considered “on.” For example, **MS_FILEHISTORY** controls whether or not MicroStation saves the last ten files and directories for each type. When this variable is set to **1**, MicroStation dialog boxes for opening files will open to the last directory accessed and file histories will appear in **File** and **Directory** pull-down menus.

Some boolean variables contain additional options, with **0** being one of the active options. For example, **MS_WORKSPACEOPTS** determines whether or not the workspace option menus are displayed and accessible in the MicroStation Manager dialog box. If set to **0** (the default), workspace options display. If set to **1**, workspace options are displayed but disabled. If set to **2**, workspace options are hidden

Configuration Variable Syntax

Configuration files are simply text files, so you can edit the configuration variables they contain in any text editor. The following syntax should be used:

<variable name> <operator> <value>

Variable names must contain at least two alphanumeric characters (**A-Z, 1-9**). Uppercase letters are used by convention.

The value assigned to a configuration variable may include the definition of another variable. For example, the variable that sets the default output directory for cell libraries could be defined as follows:

variable name	operator	value
MS_CELLOUT	=	\$_(USTN_PROJECTDATA)cell/

In this example, the value of **MS_CELLOUT** is equal to the cell directory added to the expansion of **\$_(USTN_PROJECTDATA)**. By default, **_USTN_PROJECTDATA** points to the **\Program Files\Bentley\Workspace\projects\<project name>** directory, so the value of **MS_CELLOUT** would be **\Program Files\Bentley\Workspace\projects\<project name>cell**.

Paths

Paths specified in configuration variables should use the forward slash (/) instead of the back slash (\). Also, variable definitions representing paths should have a trailing slash on the end of the definition. When defining a variable based on the expansion of another variable that includes a trailing slash, it is not necessary to include a slash after the **\$(<variable name>)** construction. For example, the default definition of **_USTN_SITE** is as follows:

_USTN_SITE = \$_(USTN_WORKSPACEROOT)standards/

Because the definition of **_USTN_SITE** includes a trailing slash, the **\$(_USTN_SITE)** construction does not require a trailing slash. For example, **MS_CELL** could be defined as follows:

MS_CELL = \$_(USTN_SITE)cell/

Notice that there is no slash after **\$(_USTN_SITE)** in the definition above.

Parentheses vs. Brackets

Variables can be defined at several different levels of priority. If a variable definition includes the name of a variable enclosed by parentheses, MicroStation will use the final definition of the variable after all processing is complete. For example, **\$(MS_CELL)** will use the final definition of **MS_CELL**, regardless of the level of the configuration file that contains the definition.

Occasionally, you may find it helpful to use the definition of a variable at a particular level of processing instead of the final definition. If so, you could enclose the name of the variable in brackets. For example, **#{MS_CELL}** refers to the definition of **MS_CELL** at the current level of processing.

Comments

When editing configuration variables, it is a good idea to add comments liberally throughout the configuration file. Comments make the configuration file easier to understand by describing the purpose of a given variable. This is especially helpful for others who may need to decipher a configuration file that you have set up. Comments begin with the pound sign (#) and can be placed on their own line or after a variable definition, as in the example below:

```
# Set the corporate settings file
MS_SETTINGS = $_USTN_SITE)data/corp.stg
```

Or...

```
MS_SETTINGS = $_USTN_SITE)data/corp.stg # Set the corporate settings file
```

Operators

Operators determine how the value of a configuration variable will be applied. The table below shows the available operators.

Operator	Meaning
=	Assigns the new value to the variable.
:	Assigns the new value to the variable only if the variable does not already exist.
+	Append string to the current value of the variable. Not intended for use with directory and file lists.
>	Append additional value to the end of a variable definition that defines a path.
<	Prepend additional value to the beginning of a variable definition that defines a path. The prepended directory is searched first.

As an example of how operators are used, assume that you are using **MS_RFDIR** to point to the directory where corporate standard reference files, such as title blocks, are located. Perhaps you have set this variable in a site-level configuration file using the value shown below:

```
MS_RFDIR = $_USTN_SITE)dgn/
```

Now assume that you are defining a project configuration. Users must still be able to locate the reference files described above, but they must also be able to locate project-specific reference files from a separate project directory. You could use the append or prepend operator to add the project directory to the search paths for reference files:

```
MS_RFDIR > $_USTN_PROJECTDATA)dgn/
```

Preprocessor Directives

Preprocessor directives control how MicroStation processes configuration variables. Preprocessor directives always begin with the percent (%) character. The table below shows the available preprocessor directives.

Operator	Meaning
%include <filename>	Process another configuration file. The filename can include references to other variables and/or the asterisk (*) wildcard character.
%if <expression>	Execute the following lines if the expression is true. Options include %if defined () and %if exists () .
%if \$(<variable name>) == "<value>"	Execute the following lines if the variable is equal to the value specified.
%else	Execute the following lines if the last %if expression was false.
%elif <expression>	Execute the following lines if the last %if was false and the expression is true.
%endif	End of conditional block.
%error <string>	Print string and exit MicroStation.
%undef <variable name>	Undefine and delete the specified variable. <variable name>=<space> defines the variable as NULL.
%lock <string>	Lock configuration variable to its current value.
%level <processing level>	Sets the level at which subsequent variables should be processed. Example: %level 2 for system level.

As an example of how preprocessor directives are used, you could use the following lines to check if a variable called **CORP_STANDARDS** has been defined and, if it has, to process any configuration files in the directory it points to:

```

%if defined (CORP_STANDARDS)           Has variable been defined?
%  if exists ($(CORP_STANDARDS)*.cfg)  If so, do .CFG files exist in directory?
%    include ($(CORP_STANDARDS)*.cfg)  If so, process .CFG files.
%  endif
%endif
    
```

Another example shows how to check the value of a variable called **CLIENT_ID**. If the value is met, then attach the following client DGN Libraries.

```

%if $(CLIENT_ID) = "MDOT"             Is CLIENT_ID defined as MDOT?
  MS_DGNLIBLIST > MDOT-Text.dgnlib    If so, attach the DGN Library?
  MS_DGNLIBLIST > MDOT-Dims.dgnlib    If so, attach the DGN Library?
%endif
    
```

Building a Corporate MicroStation Workspace

The `%lock` directive allows you to lock a configuration variable's value so that users cannot change it. For example:

```
MS_ACCUDRAWKEYS = $(<some variable>)data/shortcut.txt
%lock MS_ACCUDRAWKEYS
```

Negate Symbol

The exclamation point (!) can be used to test for a condition that is not true. For example:

```
%if !defined (CORP_STANDARDS)           Has variable NOT been defined?
%include $(SOME_VARIABLE)*.cfg         Process files from another directory.
%endif
```

Predefined Variables

Predefined variables are created during the installation of MicroStation. They determine values such as the version of MicroStation installed, the operating system, and others. They can be used to assist in processing other variables. The variables below are an example of MicroStation V8 installed on a Windows based operating system.

Variable	Default Value
<code>_VERSION80</code>	null
<code>_MICROSTATION</code>	null
<code>_ENGINENAME</code>	MicroStation
<code>_WINNT</code>	null
<code>_INTELNT</code>	null
<code>_PLATFORMNAME</code>	intelnt
<code>_WORKDIR</code>	C:\Program Files\Bentley\Program\MicroStation\
<code>_ROOTDIR</code>	C:\Program Files\Bentley\Program\MicroStation\

Logical “And” and “Or” Expressions

You can create more complicated boolean expressions using the logical “and” (`&&`) and logical “or” (`||`) symbols. For example, suppose that some users in your organization are using MicroStation V7 and others are using either MicroStation V8. To isolate a set of instructions for MicroStation V8, you could use the following expression:

```
%if defined (_VERSION80) && !defined (MS_FEATURE)
```

The `_VERSION80` variable distinguishes MicroStation V8 from MicroStation V7, but this variable is used by both MicroStation v8.0 and v8.5. The logical “and” (`&&`) allows you to test for a second condition. In this example, it tests to see whether `MS_FEATURE` is defined. This variable was added in v8.5 for feature modeling and does not exist in v8.0. Therefore, if `_VERSION80` is defined but `MS_FEATURE` is not, the version is 8.0.

Setting Up a Site Configuration

As described in the earlier, MicroStation allows you to define site-level variables that are processed by everyone at the site. These variables provide a means of implementing corporate standards. Key configuration variables are described below.

Important Configuration Variables	
<code>_USTN_WORKSPACEROOT</code>	Root directory for workspace data. Default: <code>\$(_USTN_BENTLEYROOT)Workspace/</code>
<code>_USTN_SITE</code>	Directory containing the site configuration. Default: <code>\$(_USTN_WORKSPACEROOT)standards/</code>

Segregating Workspace Data

Before developing a corporate workspace, it is important to segregate your customization from the core program. This will allow you to reinstall MicroStation without losing your customization. It will also allow you to maintain a single repository for workspace data, instead of having to maintain the data on multiple computers.

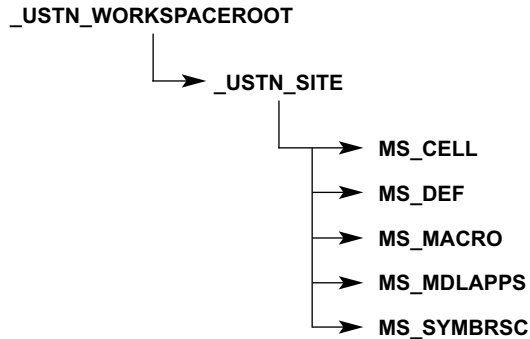
You can begin by moving all of your workspace data out of the main MicroStation directory structure. If you like, you can move or copy the entire **Workspace** directory to the new location. Ideally, the repository for workspace data should be on a network drive where you can control read/write access.

With the data in the desired network location, you can then direct MicroStation to the new root workspace directory with the `_USTN_WORKSPACEROOT` variable. By default, this variable is set in the system configuration file called `mslocal.cfg`, but you can define it in another configuration file if desired.

Building a Corporate MicroStation Workspace

By default, `_USTN_SITE` is defined as `$(_USTN_WORKSPACEROOT)standards/`. Most of the variables that point to corporate standard data are based on `_USTN_SITE`. For example, `MS_CELL` (the variable that instructs MicroStation where to look for cell libraries) is defined as `$(_USTN_SITE)cell/`. Since `_USTN_SITE` is itself based on `_USTN_WORKSPACEROOT`, you can redirect most important site variables by simply redefining `_USTN_WORKSPACEROOT`. The figure below describes this relationship graphically.

Figure 3
Setting Site Configuration Variables



Read/Write Privileges

As a general rule, workspace data should be available to users in Read-Only format. The exceptions to this rule are user configuration files and user preference files, which require both read and write access by the user. You may also wish to provide users with read/write access to personalized interfaces, function key menus, AccuDraw shortcuts, etc.



You can create other site configuration files, but they should be placed in the directory pointed to by `$(_USTN_SITE)` so that MicroStation will process them at the site-level.

The Standards.cfg Site Configuration File

`Standards.cfg` is the main site configuration file. By default, this file is located in the `\Workspace\standards` directory. Use this file to set up any variables that you want to be processed by everyone at the site. For example, you can set variables that specify the location of corporate standard cell libraries, settings files and macros. Many such variables are already set in the default `standards.cfg` file. You will likely want to modify some of the definitions and add additional variables to accommodate the requirements of your site.

Creating Project Configurations

Another important component of a corporate workspace is the project configuration. A project configuration contains project-specific variable definitions that override what is set at the site level. Key configuration variables are described below.

Important Configuration Variables	
_USTN_PROJECTSROOT	Root directory for projects. Default definition: \$_(USTN_WORKSPACEROOT)projects/
_USTN_PROJECT	Directory containing project configuration files. Default definition: \$_(USTN_PROJECTSROOT)
_USTN_PROJECTNAME	Name of current project. Default definition: Name of current project
_USTN_PROJECTDATA	Parent directory for the project data. Default definition: \$_(USTN_PROJECT)\$(USTN_PROJECTNAME)/
_USTN_OUT	Output directory for projects. Default definition: \$_(USTN_PROJECTDATA)out/



If you name some of your project data resources, such as project seed files, with the name of the project, you can create generic variable definitions based on **_USTN_PROJECTNAME** that will work with all projects. For example, you could point to a seed file called **project1seed.dgn** by defining the **MS_DESIGNSEED** as **\$_(USTN_PROJECTNAME)seed.dgn**.

Creating a Project Configuration File

MicroStation provides a template that can be used as a starting point for new project configurations. This file, called **project.template**, is located in the **\Workspace\projects** directory by default. Because it already contains many basic project variables set, you can create a copy of this file called **<project name>.pcf**, and then adjust variable definitions as needed. If you prefer, you can create a copy of the **project.template** file, and then customize it to define a corporate standard project template, from which all new project configuration files are created. With some foresight, you can design a project template that will require very little modification as new standard projects are added.

The **_USTN_PROJECTSROOT** variable points to the root directory for projects. Since this variable is based on **_USTN_WORKSPACEROOT**, MicroStation will automatically look for projects in a subdirectory of the workspace root directory, wherever that may be.

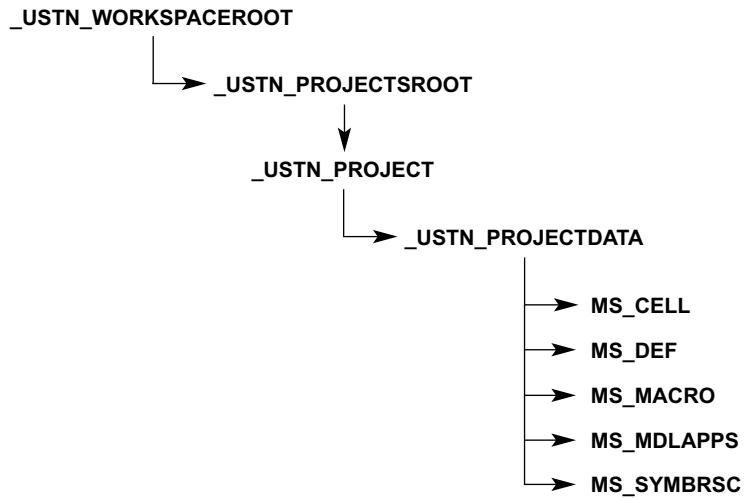
_USTN_PROJECT points to the location of project configuration files. This variable is based on **_USTN_PROJECTSROOT**.

Adding Project Data

MicroStation contains an untitled project directory that includes **cell**, **data**, **dgn**, **dgnlib**, **out**, and **seed** subdirectories. If you like, you can use this directory structure as a template for new projects by creating a copy of the directory structure in the same parent directory (**projects**, by default) and renaming it with the name of the new project. Project data, such as cell libraries, design files, etc., can then be placed in the new project's subdirectories.

The **_USTN_PROJECTDATA** variable points MicroStation to the project directory of the current project. This variable is based on **_USTN_PROJECT** and the name of the current project. When you select a project configuration file, **_USTN_PROJECTDATA** automatically directs MicroStation to the project directory with the same name, thereby making all project data available. Most project-level variables are based on **_USTN_PROJECTDATA**. The figure below describes this relationship graphically.

Figure 4
Setting Project Configuration Variables



Creating User Configurations

There are several types of user configurations. A “user” may be a specific configuration that redefines variables to point to the projects and standard data of a client, department of transportation, etc. A “user” may also be a discipline-specific configuration called “civil” or “arch,” for example. Another option is to create a “user” configuration for each MicroStation operator at a site that provides access to individual data resources. Key configuration variables are described below.

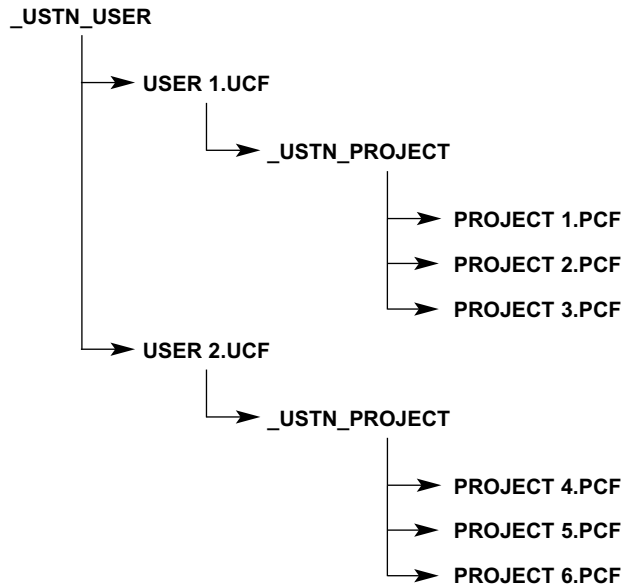
Important Configuration Variables	
_USTN_USER	Directory containing user configurations. Default: $\$(_USTN_WORKSPACEROOT)users/$
_USTN_USERNAME	Name of current user. Default definition: Name of current user

You can point to the directory that stores user configuration files (.UCF) with the **_USTN_USER** variable. Since this variable is based, by default, on the **_USTN_WORKSPACEROOT** variable, MicroStation will automatically look for user configurations in a subdirectory of the root workspace directory. The default location is **Workspace\users**.

Client and Discipline User Configurations

You can use a client- or discipline-specific user configuration to redefine the root project directory and point to projects that are specific to the client or discipline. For example, you could redefine **_USTN_PROJECT** to point to a unique set of project configuration files that are specific to the client or discipline. The figure below shows this process graphically.

Figure 5
Creating Client/Discipline-Specific User Configurations



Client and discipline user configurations can be placed in a network directory and shared by multiple users. The advantage to this approach is that the site administrator has to set up and maintain only one version of that user configuration. There is a disadvantage to sharing user configuration files, however. The last project selected by a user is recorded in the definition of **_USTN_PROJECTNAME** in the user configuration. This sets the default project and eliminates the need to reselect the same project each time MicroStation is loaded. If multiple users are sharing the same configuration file, MicroStation will no longer be able to store the last project used by a particular user.

Individual User Configurations

Often, it is preferable to set up a separate user configuration for each MicroStation operator. With this approach, each user has access to their own data resources, such as function key menus, AccuDraw shortcuts, etc. This also allows MicroStation to record the last project selected by the user and gives the administrator the ability to set up different configurations for different levels of users (beginners, power users, etc.)

In Windows, you can take advantage of the **USERNAME** environment variable to automatically point MicroStation to the data resources of a particular user based on their login ID. For example, you could create a user configuration file called **<username>.ucf** for each user, the user name being the login ID of the user. In addition, assume that you have created a user data directory somewhere on your network that contains a **<username>** subdirectory for each user. You could then incorporate **\$(USERNAME)** into the definitions of variables to point to user-specific data resources, as shown below:

```
MS_FKEYMNU = <drive:/users/$(USERNAME)/$(USERNAME).mnu
MS_ACCUDRAWKEYS = <drive:/users/$(USERNAME)/$(USERNAME).txt
```

The figure below describes this process graphically.

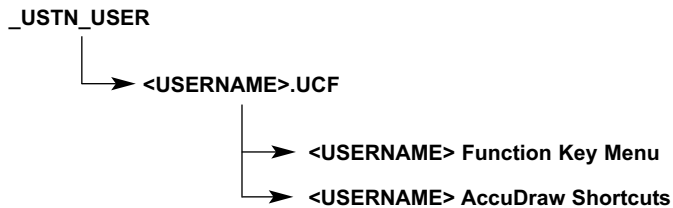


Figure 6
Accessing User-Specific Data

Isolating a User Configuration

An administrator may wish to limit which user configurations appear in the MicroStation Manager dialog box. This prevents users from selecting the wrong user configuration. It also prevents them from modifying another user's user configuration.

The MicroStation Manager dialog box displays all user configurations found in the directory pointed to by the `_USTN_USER` variable. Therefore, to isolate a user's individual user configuration, you must move it to its own directory and redirect `_USTN_USER` to the new directory. If you are using the `$(USERNAME)` Windows variable, this process is simple.



To set the default user:

1. Create a folder called `<username>` for each user. If you are accessing user configurations from a server, these user-specific folders should be stored in the same parent folder.
2. Create a `<username>.ucf` user configuration file for each user and store it in the user's `<username>` folder.
3. In a site configuration file, set `_USTN_USER` to point to the `<username>` folder and lock the definition with the `%lock` directive. For example:

```
_USTN_USER = S:/Users/$(USERNAME)/  
%lock _USTN_USER
```

4. Set `_USTN_USERNAME` to `$(USERNAME)` and lock the definition with the `%lock` directive. For example:

```
_USTN_USERNAME = $(USERNAME)  
%lock _USTN_USERNAME
```

The Default User Configuration

The default user configuration is determined by a variable called **_USTN_USERNAME**. MicroStation sets a definition of this variable in a file called **dfltuser.cfg**. This is how MicroStation “remembers” the last user configuration selected by the user. In MicroStation, **dfltuser.cfg** is stored in the directory pointed to by **_USTN_HOMEPREFS**, which is, by default, **\Program files\Bentley\Home\prefs**. (In earlier versions of MicroStation this file was stored in the directory pointed to by **_USTN_USER**) If you were to open this file, it would contain a line similar to what is shown below.

```
_USTN_USERNAME = jgreen
```

If **_USTN_USERNAME** is not defined elsewhere, MicroStation will attempt to load the user configuration specified in this file. If the user selects a different user configuration, the definition of **_USTN_USERNAME** is redefined.

If the user switches from one workspace to another, a text window may appear with a notification that MicroStation is unable to locate a particular user configuration. When this happens, MicroStation fails to load. There are two ways to address this problem. The first is to redefine **_USTN_USERNAME** in **dfltuser.cfg** to point to a user configuration in the active workspace. The second method is to set **_USTN_USERNAME** in another configuration file, such as **mslocal.cfg** or a site configuration file. The new definition of **_USTN_USERNAME** will take priority, because MicroStation only processes the definition in **dfltuser.cfg** if **_USTN_USERNAME** is not already defined elsewhere.

Setting Up a User Interface

Just as there are many different ways to set up a user configuration, there are also many different ways to set up a user interface. For example, a user interface can be discipline-specific to provide access to tools that are associated with a particular design discipline. An interface can also be client-specific to encourage compliance with a client's standards. Or you may choose to set up an individual user interface for each user, thereby allowing them to create custom tool boxes, pull-down menus, etc. The table below lists the key user interface configuration variables.

Important Configuration Variables	
_USTN_USERINTROOT	Root directory for interfaces. Default definition: \$_(USTN_WORKSPACEROOT)interfaces/
_USTN_USERINT	Parent directory of interface directories. Default: \$_(USTN_USERINTROOT)\$(ENGINE_NAME)/
_USTN_DEFUSERINTNAME	Sets the default interface if _USTN_USERINTNAME is undefined. Default: Default
_USTN_USERINTNAME	Name of active user interface. Default: Active user interface

The **_USTN_USERINTROOT** variable points to the root directory for interfaces (**Workspace\interfaces** by default). This variable then becomes the basis for the definition of **_USTN_USERINT**, which is the parent directory for directories containing user interface modification files.



The drawback to isolating each user's interface is that it takes away the ability to share tool boxes in the Customize dialog box. The **Available Tools From** option menu in this dialog box displays the available tools in all interfaces found in the directory pointed to by `_USTN_USERINT`.

Isolating a User Interface

An administrator may choose to limit which user interfaces appear in the MicroStation Manager dialog box to prevent a user from modifying another user's interface. The process for isolating the user's own interface is similar to the process of isolating the user configuration. Because the MicroStation Manager dialog box displays all interface subdirectories found in the directory pointed to by the `_USTN_USERINT` variable, you must move the user's interface folder to a different directory and redirect `_USTN_USERINT` to the new location. Following is an example workflow.



To isolate a user interface:

1. Create a folder called `<username>` for each user. If you are accessing user configurations from a server, these user-specific folders should be stored in the same parent folder. This can be the same `<username>` folder used to store the user configuration files.
2. Place the user's `<username>` interface folder containing the `ustn.m01` file in the new `<username>` folder you created.
3. In a site configuration file, set `_USTN_USERINT` to point to the first `<username>` folder and lock the definition with the `%lock` directive. For example:

```
_USTN_USERINT = S:/Users/$(USERNAME)/  
%lock _USTN_USERINT
```

4. Set `_USTN_USERINTNAME` to `$(USERNAME)` and lock the definition with the `%lock` directive. For example:

```
_USTN_USERINTNAME = $(USERNAME)  
%lock _USTN_USERINTNAME
```

Integrating a Corporate Interface

MicroStation has the ability to layer interface modification files to create a composite interface. This allows you to create a corporate standard interface, yet still allow users to customize their own user interfaces. Key interface variables are described below.

Important Configuration Variables

_USTN_UIPATH

Directory(ies) from which to read user interface modification files. Default definition:

```
$_USTN_USERINT)$_USTN_USERINTNAME)/  
$_USTN_USERINT)$_USTN_DEFUSERINTNAME)/  
$_USTN_UISTANDARDS)
```

_USTN_UISTANDARDS

Directory containing corporate standard interface modification file(s). Default definition: Undefined

The ***_USTN_UIPATH*** variable tells MicroStation from which directory(ies) to read user interface modification files. The default definition of this variable in ***msconfig.cfg*** is as follows:

```
_USTN_UIPATH      = $_USTN_USERINT)$_USTN_USERINTNAME)/  
_USTN_UIPATH      < $_USTN_USERINT)$_USTN_DEFUSERINTNAME)/  
_USTN_UIPATH      < $_USTN_UISTANDARDS)  
%lock _USTN_UIPATH
```

Notice that ***_USTN_UISTANDARDS***, the variable that points to the directory where corporate standard interface modification files are stored, is included in the search path. By default, however, this variable is undefined. You can define it in ***mslocal.cfg***, ***standards.cfg*** or elsewhere.

To prevent users from seeing the corporate standard interface as one of the options in the MicroStation Manager dialog box, ***_USTN_UISTANDARDS*** should point to a location outside of the directory pointed to by ***_USTN_USERINT***.