DPL Cookbook
Table of Contents

1 Getting Started in DPL – A Quick Tutorial ......................................................... 5
  1.1 Create a small sample project ................................................................. 5
  1.2 Create a DPL command object ............................................................... 6
  1.3 Create a filter and put it inside the DPL command object ...................... 8
  1.4 Write the DPL Script ........................................................................... 11

2 Anatomy of a DPL Object ........................................................................... 14

3 Basic DPL Scripting .................................................................................. 18
  3.1 Accessing Network Objects .................................................................. 18
     3.1.1 By the General Selection .............................................................. 18
     3.1.2 By Sets ...................................................................................... 20
     3.1.3 By Filters ................................................................................. 21
     3.1.4 By Code ................................................................................... 22
  3.2 Identifying, Accessing and Modifying Object Parameters .................. 24
     3.2.1 Identifying Variable Names for a Parameter .................................. 24
     3.2.2 Accessing Parameters .................................................................. 25
     3.2.3 Modifying Parameters .................................................................. 25
  3.3 Creating New Objects .......................................................................... 25
     3.3.1 Copy from an Internal Template .................................................. 25
     3.3.2 Copy from an External Template .................................................. 27
     3.3.3 Create a New Object by Code ....................................................... 28
  3.4 Checking if a Project is Active .............................................................. 29
  3.5 Navigating Folders and Object Contents ............................................. 29
     3.5.1 Project Folders .......................................................................... 29
     3.5.2 Object Contents ........................................................................ 30
     3.5.3 Objects in a Study Case .............................................................. 31
  3.6 Accessing Study Cases ........................................................................ 31
  3.7 Executing Calculations ....................................................................... 33
  3.8 Accessing Results .............................................................................. 34
     3.8.1 Static Calculations (Load Flow, Short Circuit, etc) ...................... 34
     3.8.2 Dynamic Simulations .................................................................. 34
  3.9 Plotting Results .................................................................................. 35
     3.9.1 Creating a New Virtual Instrument Page ..................................... 35
     3.9.2 Creating a Virtual Instrument ..................................................... 35
     3.9.3 Adding Objects and Variables to Plots ....................................... 36
     3.9.4 Plotting Example ....................................................................... 37
# Table of Contents

4 Advanced DPL Scripting ........................................................................................................... 38
  4.1 DPL Sub-Scripts (or Subroutines) ..................................................................................... 38
    4.1.1 Executing Sub-Scripts ................................................................................................. 39
    4.1.2 Passing Arguments to Sub-Scripts ............................................................................... 39
    4.1.3 Accessing Sub-Script Results ....................................................................................... 40
  4.2 Vectors, Maps and Matrices ............................................................................................... 41
  4.3 Reading from and Writing to External Files ..................................................................... 41
  4.4 Topological Searches ........................................................................................................ 41

5 Working with Results Files ..................................................................................................... 42
  5.1 Adding Results Files to the DPL Script ............................................................................ 42
  5.2 Structure of Results Files ................................................................................................. 42
  5.3 Loading a Results File into Memory ................................................................................ 43
  5.4 Getting the Relevant Column Number ............................................................................ 44
  5.5 Getting Data from the Results File .................................................................................. 44
  5.6 Getting the Time Interval Data ......................................................................................... 44
  5.7 Finding the Number of Time Intervals ............................................................................ 45
  5.8 A Simple Example ............................................................................................................ 45

6 Working with Virtual Instrument Panels .............................................................................. 47
  6.1 Introduction ....................................................................................................................... 47
    6.2 Local Title Blocks ......................................................................................................... 47
      6.2.1 Creating Local Title Blocks Manually ...................................................................... 48
      6.2.2 Creating Local Title Blocks in DPL ...................................................................... 48
      6.2.3 Title Block DPL Script Example ............................................................................. 49
    6.3 Constants in Visplots ..................................................................................................... 51
      6.3.1 Constants DPL Script Example .............................................................................. 51
    6.4 Exporting VI Panels to WMF ....................................................................................... 52
1 Getting Started in DPL – A Quick Tutorial

This is a short step-by-step tutorial that attempts to get the user started in writing and running DPL scripts. Before going through this tutorial, the user should already have a basic grasp of PowerFactory handling and performing simple tasks such as load flows and short circuit simulations.

1.1 Create a small sample project

Firstly, we will create a small sample project. Create a new project and draw the following simple 20kV network using line types from the global library:

Run a load flow to make sure the model works.
1.2 Create a DPL command object

1. Go to the scripts folder of the project library:

![DPL Tutorial folder](image)

2. Right-click anywhere in the data window and select New->Others...

![Context menu](image)

3. Select "DPL Command (ComDPL)" and press OK:
4. Call the script "DPL_Loading":

![Image of Element Selection and DPL Ladder Logic window]
1.3 Create a filter and put it inside the DPL command object

1. DPL command objects can contain other objects within it. To see the contents of a DPL command object, click on the "Contents" button:

2. We want to create a filter object. Right-click anywhere in the contents area and select New->Others... Select "General Filter (SetFilter)" and press OK.
3. Name the filter "Lines" and in the "Object Filter" option, select "Line".

![Image of General Filter window]

4. In the "Look In" option, select Network Data:

![Image of Please Select Object window]

5. Finally, check the "Include Subfolders" option. The final filter object should look like this:

![Image of the final filter object]
6. Click on the “Apply” button, and the three line objects should appear in a data window:

7. Press OK. The filter object should now be saved inside the DPL command object. When objects are saved inside the DPL command object, they can be called by a DPL script by simply referencing the name of the saved object, i.e. in this case “Lines”.

Now we will start to write the DPL script. Close the Contents window.
1.4 Write the DPL Script

1. Click on the "Script" tab in the DPL command object:

![DPL Command - Script tab highlighted]

2. We will now write a DPL script that will execute a load flow and print out the name of each line and its corresponding loading in the output window. Type the following code into the script area:

```dpl
! Variable declarations
set sLines;
object ldf, oLine;

! Run a load flow
ldf = GetCaseObject('ComLdf');
ldf.Execute();

! Get the set of lines contained in the filter
! The object "Lines" is the filter contained in the DPL command object
sLines = Lines.Get();

! Go through each line, and show name and loading
```
oLine = sLines.First();
while (oLine) {
    printf('Loading for %s: %f %%', oLine.loc_name, oLine.c.loading);
    oLine = sLines.Next();
}

The script should look like this:

![DPL Command - Scripts\DPL_loading.ComDpl](image)

3. Press "Save" and then "Execute". The following results should be seen in the output window:

```
DIGISI/info - DPL Program 'DPL_loading' started
DIGISI/info - Element 'External Grid' is local reference in separated area of '1'
DIGISI/info - Calculating load flow...
```
DIGSI/info - Start Newton-Raphson Algorithm...
DIGSI/info - load flow iteration: 1
DIGSI/info - load flow iteration: 2
DIGSI/info - Newton-Raphson converged with 2 iterations.
DIGSI/info - Load flow calculation successful.
Loading for Line1: 9,862179 %
Loading for Line2: 19,908511 %
Loading for Line3: 30,347751 %
DIGSI/info - DPL program 'DPL_Loading' successfully executed

Congratulations, you've just written your first DPL script!
2 Anatomy of a DPL Object

To understand how DPL works, it is important to understand the general structure of a complete DPL object. The actual script or code is only a part of the story, and this script is actually contained inside an object called the “DPL Command” object. Now we’ll have a look at what else is inside the DPL command object.

DPL command objects are normally located in the project library under “Scripts”:

Whenever you open a DPL command object, you will see a window with a number of tabs – Basic Options, Advanced Options, Script, Description and Version. We will go through each tab and the functions that are available.
The **Basic Options** tab (shown above) has the following functions:

- Change the name of the DPL command object
- Choose a general selection (e.g. a set) that can accessed in the script as the set object "SEL"
- Define internal input parameters – in the script, the names of the input parameters can be used like variables and do not need to be redefined
- Define external objects – these are objects external to the DPL command object that you want to access via the script. This can be any object, e.g. an element in the network model, a study case, an equipment type, etc. Inside the script, the object can be used like a variable by calling the name defined here.

The **Advanced Options** tab (shown below) has the following functions:

- Select a remote script – rather than use a script defined locally in the DPL command object, you can select a script that has been defined in a separate object.

  One reason to employ a remote script is if you have multiple study cases using the same script code, but different input parameters. If you were to use a remote script, then modifying the master script will affect all the DPL objects that refer to it. If you had locally defined scripts, then you would need to change the code in every single local DPL object individually.

- Define result parameters – these are essentially output parameters that are stored inside the DPL command object (even after script execution). You can access these parameters as if they were from any other type of object, i.e. either by another DPL script or as a variable to display on a plot.
The **Script** tab (shown below) is where you write the script code.

The **Description** and **Version** tabs are informational tabs for script descriptions and revision information.
Lastly, the DPL command object is also a container for other objects. By clicking on the “Contents” button, you can see what is inside the DPL command object:

Inside the DPL command object above, you can see that it contains a vector object “dpl_vec” and another DPL command object “sub_max_vector” (this is a sub-script – see Section 4.1 for further details). Much like input parameters and external objects, the objects contained in the DPL command object can be accessed inside the script like variables with the names of the object (they do not need to be re-defined in the script).
3 Basic DPL Scripting

3.1 Accessing Network Objects

There are four main approaches to accessing network objects via DPL:

- By the general selection
- By sets
- By filters
- By code (internal functions)

3.1.1 By the General Selection

The general selection is a set that can be defined in the Basic Options tab:
Alternatively, the general selection can be defined by selecting elements in the single line diagram and pressing right-click -> Execute DPL Scripts (see the figure below).

Inside the script, the general selection can be accessed by invoking the special reserved object variable "SEL". For example, the code snippet below gets all of the lines in the general selection:

```plaintext
set sLines;

! Get the set of lines contained in the general selection
sLines = SEL.AllLines();
```
3.1.2 By Sets

Sets can be included inside the DPL command object or referenced as an external object. For example, the DPL command object below contains the set object “Set”.

Inside the script, this set can be accessed using the name “Set”. For example, the code snippet below goes through each object in the set object “Set” and prints out its full name.

```plaintext
object oObj;

! Cycle through the objects in the set and print out the full name
oObj = Set.First();
while(oObj) {
```

DPL Cookbook 20
oObj.ShowFullName();
oObj = Set.Next();
}

3.1.3 By Filters

Filter objects have to be put inside the DPL command object for them to be accessible by DPL scripts.

To see what’s inside a DPL command object, press the “Contents” button on the right-hand side of the DPL command dialog box.

![DPL Command dialog box with Contents button highlighted]

Firstly, create a filter inside the DPL command object. For example the filter “Lines”, which finds all of the lines in the project, is created in the DPL command object below:
The name of the filter object will be the name of that is used when referencing the object from inside the DPL script. Within the code, use the `Get()` command to obtain a set from the filter. For example, the code snippet below gets the results from the filter "Lines" and puts it into the set "sLines":

```plaintext
set sLines;

! Get the set of lines contained in the filter "Lines"
! The object "Lines" is the filter contained in the DPL command object
sLines = Lines.Get();
```

### 3.1.4 By Code

The general approach for accessing network objects in DPL purely through code is as follows:

1. Get a set of the relevant network objects that you are looking for (using the "AllRelevant" command), based on a specific element type, e.g. lines, transformers, motors, etc
2. Get an object within this set either through filter commands (e.g. FirstFilt, NextFilt, etc) or by order commands (e.g. First, Next, etc)

The code snippet below gets the set of all lines, cycles through each line object and prints out its full name:

```plaintext
object aLine;
set Lines;

! Get the set of all lines
Lines = AllRelevant('*.ElmLine', 1, 1);

! Cycle through the lines and print out the full name
aLine = Lines.First();
while(aLine) {
```
aLine.ShowFullName();
aLine = Lines.Next();
}

The code snippet below gets the set of all objects, and tries to find the particular line called "LineA5":

object aLine;
set    AllObjs;

! Get the set of all objects
AllObjs = AllRelevant();

! Get the line called "LineA5" and print out its full name
aLine = AllObjs.FirstFilt('LineA5.ElmLine');
aLine.ShowFullName();

The code snippet below gets the set of all objects, filters for all lines starting with "LineA2", cycles through each line object in the filtered and prints out its full name.

object aLine;
set    AllObjs;

! Get the set of all objects
AllObjs = AllRelevant();

! Filter the set with all lines starting with "LineA2"
aLine = AllObjs.FirstFilt('LineA2*.ElmLine');

! Go through filtered set and print out the full name of each object
while(aLine) {
    aLine.ShowFullName();
    aLine = AllObjs.NextFilt();
}
3.2 Identifying, Accessing and Modifying Object Parameters

Once a specific object has been selected, the way to access the object parameters or variables is by typing out the object name and the variable name separated by a colon (\text{:}), e.g.

\text{Object\_name:Variable\_name;}

3.2.1 Identifying Variable Names for a Parameter

Most of the time it isn’t obvious what the variable name for a particular object parameter is. For example, suppose you want to access the power factor parameter in a static generator element — what is the variable name?

Variable names can often be found in the manual and in the technical references, but the easiest way to identify variable names is to go into the relevant object and hover the mouse over the field of interest. A tooltip will appear with the corresponding variable name. For example, hovering over the power factor field in the static generator element yields the variable name: \text{“cosn”}:
3.2.2 Accessing Parameters

Suppose we have a line object \texttt{沿线} and we want to save the length of the line (variable name = \texttt{dline}) to an internal DPL variable \texttt{dLength}, this is how to do it:

\[ \texttt{dLength =沿线:dline;} \]

3.2.3 Modifying Parameters

Suppose we have a line object \texttt{沿线} and we want to change the length of the line (variable name = \texttt{dline}) to 2km, this is how to do it:

\[ \texttt{沿线:dline} = 2; \]

3.3 Creating New Objects

There are at least three ways to create new objects in the database:

1. Copy template object contained inside the script object
2. Copy external template object
3. Create new object from scratch in code

3.3.1 Copy from an Internal Template

Copying an internal template object is potentially the easiest option for creating new objects. The DPL command object can contain other objects within it. To see what's inside a DPL object, click on the Contents button of the DPL command dialog box:
You can use an internal object inside the DPL command object as a template for the object type you want to create. In the figure above, an event (EvtShc) object call "SC_Event" is placed inside the DPL command object and used as the template.

Now to create the object, you can use the **AddCopy(object_name)** command. In order to use this command, you must first locate the folder (or object) that you want to copy the template into.

For example, suppose "oFold" is an object with the reference to the target folder and "SC_Event" is the internal template object, a new object will be created in the target folder by the following:

```plaintext
oFold.AddCopy(SC_Event);
```

The code snippet below copies an internal short circuit event template into the simulation events folder of the active study case and then changes the time of the copied event to t=1.

```plaintext
object aPrj,
   oFold,
   oEvent;
set EventSet;

! Get the simulation events folder in the active study case
```
oFold = GetCaseObject('IntEvt');
oFold.ShowFullName();

! Copy the template short circuit event into the events folder
oFold.AddCopy(SC_Event);
! Get the copied event and set the time to 1
EventSet = oFold.GetContents('@.EvtShc');
oEvent = EventSet.First();
oEvent:time = 1;

3.3.2 Copy from an External Template

This is almost identical to copying an object from an internal template, except that an external object is referenced instead of using an object from inside the DPL command. In the DPL command dialog (under "Basic Options"), there is an area for referencing external objects:
To use this area, simply double-click on the object field and select an object you'd like to reference from pretty much anywhere in the project hierarchy. One distinction of this method is that you can give the object any name that you like (using the "Name" field). This is the name that the object will be called inside the DPL script.

Once the external template object has been selected, you can use the `AddCopy(object_name)` command to create a new object in the database. In order to use this command, you must first locate the folder (or object) that you want to copy the template into.

For example, suppose "oFold" is an object with the reference to the target folder and "Trig" is the external template object, a new object will be created in the target folder by the following:

```plaintext
oFold.AddCopy(Trig);
```

Refer to the example given earlier in Section 3.3.1 for more details.

### 3.3.3 Create a New Object by Code

Creating new objects purely by code is the most intensive method for making new objects, because all of the object parameters need to be set in code. With template objects, you can set default parameters and even add child objects inside the template. But creating objects purely in code requires all of this to be done manually in the script.

The `CreateObject(class_name, object_name)` function is used to create new objects. Like the previous two methods, you must first locate the folder (or object) that you want to create the object in.

For example, suppose "oFold" is an object with the reference to the target folder and you want to create a short-circuit event object (`.EvtShc`) called "SC_Event", you would use the following command:

```plaintext
oFold.CreateObject('EvtShc', 'SC_Event');
```

Note that if the target folder (or object) does not accept the object class you are trying to create (e.g. a VI page object in the simulation events folder), then an error will be raised.

The code snippet below creates a new short circuit event into the simulation events folder of the active study case and then sets the time of the event to t=1.

```plaintext
object aPrj, oFold, oEvent;
set EventSet;

! Get the simulation events folder in the active study case
oFold = GetCaseObject('IntEvt');
oFold.ShowFullName();
```
Create short circuit event into the events folder
oFold.CreateObject('EvtShc', 'SC_Event');

Get the copied event and set the time to 1
EventSet = oFold.GetContents('*EvtShc');
oEvent = EventSet.First();
oEvent:time = 1;

3.4 Checking if a Project is Active

The function ActiveProject() returns NULL if there no project currently active.

object aPrj;
! Check if there is an active project
aPrj = ActiveProject();
if (aPrj=NULL) {
    Error('Please activate a project first');
    exit();
}

3.5 Navigating Folders and Object Contents

3.5.1 Project Folders

As an entry or starting point into the project folder hierarchy, use the GetProjectFolder(string) function. This function will return an object with a reference to the top level of project folders, e.g. the folders containing study cases, scripts, libraries, diagrams, etc.

For example, the following line puts a reference to the equipment type library folder into object "oFold":

    oFold = GetProjectFolder("equip");

A list of the project folders available and the corresponding string is shown below:

<table>
<thead>
<tr>
<th>String</th>
<th>Folder Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>equip</td>
<td>Equipment type library</td>
</tr>
</tbody>
</table>
### 3.5.2 Object Contents

The `GetContents(string)` function is a generic way of navigating through objects and finding the set of objects contained within them. The function returns a set of objects that meet the criteria in the string.

Some examples:

1. Return all objects contained in "oObj" into the set "Contents":
   ```
   Contents = oObj.GetContents();
   ```

2. Return "ElmTerm" type objects (terminals) contained in "oObj" into the set "Contents"
   ```
   Contents = oObj.GetContents('*.ElmTerm');
   ```

3. Return the specific object "T2.ElmTerm" contained in "oObj" into the set "Contents":
   ```
   Contents = oObj.GetContents('T2.ElmTerm');
   ```

4. Return all "ElmTerm" type objects that have names starting with "T" contained in "oObj" into the set "Contents":
   ```
   Contents = oObj.GetContents('T*.ElmTerm');
   ```
3.5.3 Objects in a Study Case

In order to access objects within the active study case (e.g. calculation command objects, simulation events, graphics boards, sets, outputs of results, title blocks, etc), you can use the function `GetCaseObject(string)`.

This function is essentially a shortcut to accessing objects inside a study case, which is used in lieu of navigating to the study case project folder, selecting a study case and then selecting an object. The other advantage of `GetCaseObject()` is that if the object doesn’t exist inside the study case, the function will create it.

Note however that this function only works with the active study case.

The code snippet below gets the graphics board object from the active study case.

```plaintext
obj oDesk;
oDesk = GetCaseObject('SetDesktop');
```

3.6 Accessing Study Cases

Study cases are `IntCase` objects that are stored in the study cases project folder. In order to access a study case, you must first access the study case folder.

The code snippet below does the following:

1. Gets the set of all study cases
2. Counts the number of study cases
3. Activates each study case and prints its full name

```plaintext
object aFold, aCase;
set aCases;
int iCases;

! Get the study cases folder and make a set of all study cases
aFold = GetProjectFolder('study');
aCases = aFold.GetFiles("*.IntCase");

! Count the number of study cases
iCases = aCases.Count();
if (iCases=0) {
    Error('There are no study cases in the current selection');
    exit();
}
printf('Number of cases: %d', iCases);
```
! Cycle through all study cases
! Activate each study case and then print out the name of the study case
aCase = aCases.First();
while (aCase) {
  ! Show name of calculation case
  aCase.ShowFullName();
  aCase.Activate();
  aCase = aCases.Next();
}

Note that if there are subfolders within the study case folder, then further processing needs to be done to access the study case objects within these subfolders. Refer to Navigating Project Folders in section 3.5.
3.7 Executing Calculations

The **GetCaseObject**(string) can be used to get an existing or create a new calculation command object. The **Execute()** object function can then be used to execute the calculation. The following list of calculation command objects are available:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComLdf</td>
<td>Load flow</td>
</tr>
<tr>
<td>ComShc</td>
<td>Short circuit</td>
</tr>
<tr>
<td>ComSim</td>
<td>Time domain (RMS or EMT) simulation</td>
</tr>
<tr>
<td>ComInc</td>
<td>Time domain initial conditions</td>
</tr>
<tr>
<td>ComSimOutage</td>
<td>Contingency analysis</td>
</tr>
<tr>
<td>ComRel3</td>
<td>Reliability assessment</td>
</tr>
<tr>
<td>ComMod</td>
<td>Modal analysis</td>
</tr>
<tr>
<td>ComHlfd</td>
<td>Harmonic load flow</td>
</tr>
<tr>
<td>ComGenrelinc</td>
<td>Initialise generation adequacy</td>
</tr>
<tr>
<td>ComGenrel</td>
<td>Run generation adequacy</td>
</tr>
<tr>
<td>ComCapo</td>
<td>Optimal capacitor placement</td>
</tr>
<tr>
<td>ComVstab</td>
<td>Load flow sensitivities</td>
</tr>
<tr>
<td>ComRed</td>
<td>Network reduction</td>
</tr>
<tr>
<td>ComVsat</td>
<td>Voltage sag table assessment</td>
</tr>
<tr>
<td>ComCabsise</td>
<td>Cable reinforcement optimisation</td>
</tr>
<tr>
<td>ComTieopt</td>
<td>Tie open point optimisation</td>
</tr>
<tr>
<td>ComSe</td>
<td>State estimator</td>
</tr>
<tr>
<td>ComFlickermeter</td>
<td>Flickermeter</td>
</tr>
</tbody>
</table>

The code snippet below executes a load flow in the active study case:

```plaintext
object ldf;

! Get load flow object from calculation case (or create it)
ldf = GetCaseObject('ComLdf');

! Execute load flow calculation
ldf.Execute();
```
3.8 Accessing Results

3.8.1 Static Calculations (Load Flow, Short Circuit, etc)

The results of static calculations are stored as parameters in the objects themselves. Therefore, static results can be accessed in the same way object variables are accessed, e.g.

```
Object_name:Result_variable_name;
```

For example, suppose you had a bus object "Bus1" and you wanted to save the per-unit voltage to an internal DPL variable called "dVoltage", you would type in this:

```
dVoltage = Bus1.m:u;
```

The simple example below runs a load flow for the active study case, gets all the lines and prints out the name and loading of each line (the reference to the line loading is shown below in bold).

```
object ldf, aLine;
set Lines;

! Run a load flow
ldf = GetCaseObject('ComLdf');
ldf.Execute();

! Get the set of all lines
Lines = AllRelevant('ElemLine', 1, 1);
aLine = Lines.First();

! Go through each line, and show name and loading
while (aLine) {
    printf('Loading for %s: %6f %%', aLine.loc_name, aLine.c:loading);
    aLine = Lines.Next();
}
```

3.8.2 Dynamic Simulations

Unlike the static cases, the results of dynamic simulations are not stored as part of the object parameters, but in a separate results file. Refer to the section Working with Results Files for more information.
3.9 Plotting Results

Plots from dynamic (time-domain) or static simulations can be easily created using DPL.

3.9.1 Creating a New Virtual instrument Page

You can create a new virtual instrument (VI) page using a graphics board object (*.SetDesktop). The function 
**GetPage (string name, int create)** will create a new page provided the name doesn't refer to an existing VI 
page and the create flag is activated (=1).

For example, the snippet below uses the graphics board object oGrb to create a VI page called "Plots":

```c
object oViPg;
oViPg = oGrb.GetPage('Plots',1);
```

3.9.2 Creating a Virtual Instrument

Similar to creating a new VI page, you can use the VI page object (*.SetVipage) to create a new virtual 
instrument with the function **GetVI (string name, string class, int create)**. To create a new VI, the name 
should not be the same as another existing VI and the create flag should be activated (=1).

The class of the VI is the type of VI that you want to create. If it is not entered, the class is set as a subplot 
(*.VisPlot) by default. The classes for virtual instruments are shown in the table below:

<table>
<thead>
<tr>
<th>VI Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VisPlot</td>
<td>Subplot</td>
</tr>
<tr>
<td>VisPlot2</td>
<td>Subplot with two y-axes</td>
</tr>
<tr>
<td>VisXyplot</td>
<td>X-Y plot</td>
</tr>
<tr>
<td>VisFft</td>
<td>FFT plot</td>
</tr>
<tr>
<td>VisOcplot</td>
<td>Time-overcurrent plot</td>
</tr>
<tr>
<td>VisDraw</td>
<td>R-X plot</td>
</tr>
<tr>
<td>VisPlotz</td>
<td>Time-distance plot</td>
</tr>
<tr>
<td>VisEigen</td>
<td>Eigenvalue plot</td>
</tr>
<tr>
<td>VisModbar</td>
<td>Mode bar plot</td>
</tr>
<tr>
<td>VisModphasor</td>
<td>Mode phasor plot</td>
</tr>
<tr>
<td>VisVec</td>
<td>Vector diagram</td>
</tr>
<tr>
<td>VisHrm</td>
<td>Waveform plot</td>
</tr>
<tr>
<td>VisBdia</td>
<td>Distortion plot</td>
</tr>
<tr>
<td>VisPath</td>
<td>Voltage profile plot</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>VisVsag</td>
<td>Voltage sag plot</td>
</tr>
<tr>
<td>VisPdc</td>
<td>Probability density plot</td>
</tr>
<tr>
<td>VisDefcrv</td>
<td>Curve-input plot</td>
</tr>
<tr>
<td>VisMeas</td>
<td>Measurement (scales, displays, etc)</td>
</tr>
<tr>
<td>VisBitmap</td>
<td>Picture box</td>
</tr>
<tr>
<td>VisSwitch</td>
<td>Button</td>
</tr>
<tr>
<td>VisButton</td>
<td>Command button</td>
</tr>
</tbody>
</table>

For example, the snippet below uses the VI page object “oViPg” and creates a subplot type VI called "Subplot":

```javascript
object oPlot;
oPlot = oViPg.GetVI('Subplot','VisPlot',1);
```

### 3.9.3 Adding Objects and Variables to Plots

In order to show the actual plots on a VI, you need to add objects and variables to the VI. Using the VI object, use the `AddVars` function to add variables and objects to the plot. The function `AddVars` can be used in one of two ways:

1) **AddVars** (string \( V \), object \( O1, \ldots, O8 \))

   Here we want to add the variable \( V \) to objects \( O1 \) to \( O8 \) (i.e. variables can be added to up to 8 objects simultaneously).

   For example, the snippet below adds the variable "m:u1" for the objects "oBus1" and "oBus2" to the VI object "oPlot" (oBus1 and oBus2 could be substation buses and we want to plot the voltage "m:u1" on these buses):

   ```javascript
   oPlot.AddVars('m:u1', oBus1, oBus2);
   ```

2) **AddVars** (string \( O \), object \( V1, \ldots, V8 \))

   Alternatively, we can add multiple variables \( V1 \) to \( V8 \) to a single object \( O \) (i.e. up to 8 variables can be added to an object simultaneously).

   For example, the snippet below adds the variables "m:phiu" for the object "oBus" to the VI object "oPlot":

   ```javascript
   oPlot.AddVars(oBus, 'm:u1', 'm:phiu');
   ```
3.9.4 Plotting Example

The example below gets the current graphics board, creates a new VI page called "Plots", creates a subplot, adds the variable "m:u" for the object "oBus" (which is defined elsewhere) and then adjusts the y-axis max and min settings:

```
object oGrb, oViPg, oPlot;
set Lines;

! Get current graphics board
oGrb = GetGraphBoard();
if (oGrb=NULL) [exit(); ]

! Create VI page
oViPg = oGrb.GetPage('Plots',1);

! Create a new subplot
oPlot = oViPg.GetVI('Subplot','VisPlot',1);

! Add variable 'm:u' for object 'oBus' to the subplot
! Note that by default, the "All Calculations" results object is used
oPlot.AddVars('m:u', oBus);

! Adjust y-axis min and max settings
oPlot:y_max = 2;
oPlot:y_min = 0;
```

For further details on manipulating Virtual Instrument panels in DPL, refer to the section Working with Virtual Instrument Panels.
4 Advanced DPL Scripting

4.1 DPL Sub-Scripts (or Subroutines)

DPL scripts are able to call other DPL scripts that are contained inside the DPL command object.

For example, the DPL command object in the figure below ("max_dpl_vector") contains another DPL script called "sub_max_vector". This script ("sub_max_vector") is referred to as a sub-script or a subroutine and can be executed from inside the parent script ("max_dpl_vector").

The use of sub-scripts is quite straightforward and will be described briefly in this section.
4.1.1 Executing Sub-Scripts

To execute a sub-script, simply call the sub-script object in the code and use the `Execute()` method:

```plaintext
sub_script.Execute();
```

For example, to execute the sub-script "sub_max_vector" in the figure above:

```plaintext
sub_max_vector.Execute();
```

4.1.2 Passing Arguments to Sub-Scripts

Sub-scripts may have input parameters that need to be initialised or adjusted before execution. Consider the script "calc_power" below, which has two input parameters P and Q.
There are two general approaches for passing arguments to a sub-script within DPL:

1) Explicit Approach

The sub-script is treated as an object and the object parameters are changed directly inside the object. For example, to change the parameters P and Q in the sub-script "calc_power" above:

```cpp
calc_power:P = 4;
calc_power:Q = 3;
```

2) Implicit Approach

Alternatively, the arguments can be passed implicitly as part of the `Execute(arguments)` method, where the arguments appear in the same order as the input parameters. For example, to pass P=4 and Q=3 into the sub-script "calc_power" above:

```cpp
calc_power.Execute(4, 3);
```

4.1.3 Accessing Sub-Script Results

A sub-script may simply execute some actions (e.g. activating study cases, changing model parameters, etc), but sub-scripts may also produce results as either variables or objects. These results are defined as parameters in the "Advanced Options" tab of the DPL command object. Consider the results parameter "S" defined in the script "calc_power" below:

![Screenshot of DPL Command - Scripts\calculate\calc_power.ComDpl interface showing result parameters: Type double, Name S, Unit kVA, Description Apparent power]
Once the script is executed, the result parameter "S" is stored in the script object and can be accessed like any other object parameter, e.g.

```java
    double app_power;
    app_power = calc_power:S;
```

This is the explicit approach to accessing results from a sub-script object.

However, like passing arguments to a sub-script, there is also an implicit approach using the `Execute(arguments, results)` method. In the implicit approach, the execute command is used with the sub-script arguments (input parameters) first and then variables to store the results.

This is best illustrated by example. In the snippet below, we pass the input arguments P=4 and Q=3 and then store the results in the variable "app_power" (of type double):

```java
    double app_power;
    calc_power.Execute(4, 3, app_power);
```

### 4.2 Vectors, Maps and Matrices

Vectors, maps and matrices can be created and manipulated in one of four types:

- `IntVec`
- `IntMat`
- `IntDblvector`
- `IntDblmap`

TBA – under construction

### 4.3 Reading from and Writing to External Files

TBA – under construction

- `WriteWMF`
- `File object methods (fopen, fprintf, fscanf, etc)`

### 4.4 Topological Searches

TBA – under construction
5 Working with Results Files

Suppose you want to get a PowerFactory Results file (of type ElmRes) from a dynamic simulation, peer into it, pull out a set of relevant values, perform some calculations on the data and generate some outputs. How do you do that? This document describes how to use a DPL script to manipulate Results files from dynamic simulations.

5.1 Adding Results Files to the DPL Script

The easiest way to work with a results file is to import it into the DPL script. For example, you can drag and drop an "All Calculations" results file into your script. Note that the name of the results file needs to be changed to eliminate spaces, e.g. in the screenshot below, "All Calculations" has been renamed to "All_calcs".

Once it has been dragged into the script, the Results file can be called inside the DPL script like an object variable that has already been defined. All object methods and functions are exposed.

5.2 Structure of Results Files

In order to manipulate the data in a Results file, it is important to understand the structure of the file and how data is stored inside it. The results file is structured as a 2d matrix as shown in the figure below.
The number of rows represents the total number of time intervals for the simulation. For example, if there is 10s of simulation with a time interval of 0.01s, then there would be 1,000 rows. Each row represents a time interval.

Column 1 in the Results file is the time of the simulation.

Columns 2 onward represent the objects (shown in the diagram as object 1 to object n) and their respective variables (shown in the diagram as var1 to varn). An object can be any PowerFactory object (e.g. a line, motor, terminal, load, etc) and a variable is any element defined in a variable set for the specific object that is to be measured during the simulation (e.g. m:ui, s:speed, etc)

Note that in the diagram above, each object has k variables, but this is not necessarily the case in practice as each object can be defined with an arbitrary number of variables in the set.

It is important to know that accessing data from a specific object and variable in the Results file hinges on knowing the relevant column index. Similarly, accessing data for a particular time requires the row number for the relevant time interval.

5.3 Loading a Results File into Memory

Adding a Results file to a DPL script does not enable access to the data contained within it. You must first load the results file into memory using the command:

```plaintext
void LoadResData(ResultsFile);
```

LoadResData loads the contents of a Results file into memory and allows you to begin accessing the data. You can only load one Result file into memory at any one time.
5.4 Getting the Relevant Column Number

In order to access the object and variable of interest, you need to know the right column number in the Results file. To do this, use the following command to search for the relevant column number:

```c
int ResIndex(ResultsFile, object, 'var_name');
```

ResIndex returns an integer, which is the column number for the object and variable you've searched for. "Object" is the specific PowerFactory object of interest (which can be defined explicitly or by a general selection). "var_name" is the variable of interest (e.g. m:u1, s:speed, etc).

5.5 Getting Data from the Results File

Once you know the relevant column index (object / variable) and row index (time), you can start getting data from the Results file. Data points can only be accessed one at a time (i.e. you can't get a vector of data from the Results file). Use the following command to get a data point:

```c
void GetResData(output, ResultsFile, row index, column index);
```

The row and column indices are integers. "Output" is a variable where the data from the Results file is stored. It must be a variable that has been already defined.

5.6 Getting the Time Interval Data

Similar to getting data from the Results file, the time interval data (e.g. t=0s, t=0.5s, etc) can be accessed using GetResData, but without inputting the column index, i.e.:

```c
void GetResData(output, ResultsFile, row index);
```

where "Output" is the variable to store the time interval data.
5.7 Finding the Number of Time Intervals

To find the number of time intervals in the simulation (i.e. number of rows), use the command:

```c
int ResNVal(ResultsFile, ColumnNo);
```

The ResNVal function returns an integer with the number of time intervals for a specific column. Using ColumnNo = 0 gives the total number of rows.

5.8 A Simple Example

This example opens a Results file, selects the first object from the general selection and prints out the voltage at each time interval in the output window. Note that in the example below, All_calcs is the name of the Results file that has been added into the DPL script.

```dpl
! Define and initialise variables
object oSet;
int N, i, il;
double dt, du;
set S;
i = 1;

! From all objects in the general selection, get the first object
S = SEL.All();
oSet = S.First();

! Load the Results file "All_calcs" into memory
LoadResData(All_calcs);

! Return the number of rows (i.e. time intervals)
N = ResNVal(All_calcs, 0);

! Get the column index for the voltage 'm:ui' for the selected
! object
il = ResIndex(All_calcs, oSet, 'm:ui');

! Loop through each row and output the time and voltage
while (i<N) {
   GetData(dt, All_calcs, i);
   GetData(du, All_calcs, i, il);
}
```
if (i>1) {
    printf('t = %f s; u = %f', dt, du);
}

i = i + 1;
}
6 Working with Virtual Instrument Panels

6.1 Introduction

Virtual instrument (VI) panels are contained in "SetVipage" objects within a graphics board, and are used to visualise the results of dynamic simulations for relevant variables of interest.

This document provides some detailed instructions on manipulating virtual instrument panel objects using the DPL scripting language. Firstly some general definitions for the objects related to virtual instrument panels:

- **Graphics Board** – is the container for all graphics, including the VI panels
- **VI Panel** – is the name of the whole virtual instrument page (*.SetVipage)
- **Visplots** – are the individual plots inside a VI panel (*.VisPlot)

![Diagram of VI Panel and Visplots]

6.2 Local Title Blocks

Title blocks are "SetTitlm" objects that can be attached to VI graphical objects. The title blocks provide drawing information, revision numbers, dates and logos, and is therefore primarily used for printouts of VI graphics. One can manipulate the data in title blocks using DPL, which makes automation possible for a large number of similarly themed drawings.
6.2.1 Creating Local Title Blocks Manually

By default, a global title block is applied to each VI panel with common title block information (but with different page numbers). But sometimes you may want to have different title block information for each VI page. Local title blocks can be created with information specific to the individual VI panel.

You can create local title blocks manually by right-clicking on the default title block and selected "Create local Title" from the context menu (see screenshot below).

6.2.2 Creating Local Title Blocks in DPL

Creating local title blocks manually can be tedious if you have to do it for many VI panels. A faster way is to use a DPL script. However there isn’t an easy direct way to create local title block objects via a DPL script. Instead, the simplest way is to copy a template title block into a VI panel object.

Firstly, find the template title block and add it as an external object to the DPL command object. The template title block can be any title block in the graphics board. In the example below, the default global title block for the graphics board is used as the template (which can be found in the main graphics board object).
Next, get the VI panel object (*.SetVipage) and use the general object method `AddCopy` to copy the title block template into the object. This creates a local title block for the VI panel.

You can now fill in the title block information by getting the title block object (*.SetTitm) and accessing its variables. For example, if the title block object is `oTitle`, then you can change the annex, title 1 and title 2 by accessing the variables `oTitle:annex`, `oTitle:sub1z` and `oTitle:sub2z` respectively.

The variable names for each title block field is shown below:

![Title Block Variables](image)

### 6.2.3 Title Block DPL Script Example

This DPL script gets the first graphics board and cycles through each VI panel. The script then checks to see if it has a local title block and adds one if it doesn’t. Lastly, the script fills in the title block information. In this example, the object "CopyTemplate" is the title block template that has been added as an external object.

```dpl
object oCase, oTitle, GrBrd, Pg;
set allDesks, allTitms, Pgs;
string strStation, strVIPage;
int n;

! Get the first graphics board
oCase = ActiveCase();
allDesks = oCase.GetContents("*.SetDesktop");
GrBrd = allDesks.First();
```
if (GrBrd) {

    ! Get a set of all VI panels
    Pgs = GrBrd.GetContents("*.SetVipage");
    Pgs.SortToVar(0,'order');
    Pg = Pgs.First();

    ! Go through each VI panel
    while (Pg) {
        printf('Name of object: %s', Pg.GetFullName());

        ! Get the first local title block
        allTitms = Pg.GetContents("*.SetTitm");
        oTitle = allTitms.First();

        ! If no local title block for VI panel exists, create one
        if (.not. oTitle) {
            printf('No local title block exists... Creating local title block...
';
            Pg.AddCopy(CopyTemplate);
            allTitms = Pg.GetContents("*.SetTitm");
            oTitle = allTitms.First();
        }

        ! Fill in title block with the following information:
        ! Annex = "1"
        ! Title 1 = "Motor Re-Acceleration Simulation"
        ! Title 2 = "Voltage Profile for Station X"
        ! where X is the name of the VI panel

        oTitle:annex = '1';
        oTitle:subl = sprintf('Motor Re-Acceleration Simulation');
        strVipage = Pg:loc_name;
        oTitle:sub2 = sprintf('Voltage Profile Station %s', strVipage);

        Pg = Pgs.Next();
    }
}

**IMPORTANT NOTE:** Objects cannot be copied into a VI panel object while the graphic is open. Therefore, close the graphics board prior to executing any DPL scripts that will manipulate title block objects.
6.3 Constants in Visplots

Horizontal and vertical constants in Visplots are contained in "VisXvalue" and "VisYvalue" objects respectively. There is no easy direct way to create these objects. It is easier to copy and paste a template object rather than creating a new object from scratch and setting up the attributes to be a VisXvalue or VisYvalue object.

6.3.1 Constants DPL Script Example

This DPL script gets the first VI panel and cycles through all the Visplots in the panel. In each Visplot, it changes the colour of the first constant to red (color 2) and adds two more constants.

```dpl
object oCase, oPlot, oConstant, GrBrd, Pg;
set allDesks, allPlots, allConstants, Pgs;
int n;

! Get graphics board
oCase = ActiveCase();
allDesks = oCase.GetContents('*.SetDesktop');
GrBrd = allDesks.First();

if (GrBrd) {
    ! Get all VI panels
    Pgs = GrBrd.GetContents('*.SetVipage');
Pgs.SortToVar(0, 'order');
Pg = Pgs.First();

    ! Get the set of all Visplots in VI panel
    allPlots = Pg.GetContents('*.VisPlot');
oPlot = allPlots.First();

    ! Cycle through each Visplot
    while (oPlot) {
        ! Change colour of first constant to red
        allConstants = oPlot.GetContents('*.VisXvalue');
oConstant = allConstants.First();
```
6.4 Exporting VI Panels to WMF

Often you will want to send VI panels as a picture file or insert it into a report. Obviously, you could do this manually using the export command (i.e. to a WMF or a BMP):

But this can be a tedious task if there are many VI panels to export. A quicker alternative is to use a DPL script to bulk export VI panels or any other type of graphic. The general process is as follows:

1. Locate the relevant graphics board
2. Select the VI panels or graphic objects that you want to export
3. Export the VI panel with the WriteWMF function
The following example gets the first graphic board in the active study case and exports all of the VI panels in the graphics board as File1.wmf, File2.wmf, File3.wmf, etc in the directory strPath (a variable defined in the DPL command object):

```dpl
obj oDesk, oVIpg;
set allVis;
string strFilename;
int i, j;
i = 0;

! Get the graphics board in the active study case
oDesk = GetCaseObject('SetDesktop');

! Get the set of all VI panels and cycle through each VI panel
allVis = oDesk.GetContents('*SetVipage');
for(oVIpg = allVis.First(); oVIpg; oVIpg = allVis.Next()) {

    ! Show the VI panel onscreen
    oDesk.Show(oVIpg);

    ! Formulate the filename for the WMF file
    i = i + 1;
    strFilename = sprintf('%s\File%d', strPath, i);

    ! Export the current VI panel to a WMF
    ! If successful, output a message saying that a file is created
    j = oDesk.WriteWMF(FileName);
    if (j = 1) {
        printf('File created: %s', strFilename);
    }
}
```
Alternatively, in lieu of getting the graphics board and getting a set of the VI pages, you could use a set of the graphical objects that you want to export, e.g.