Procedure to convert CONEC and CONET type models to the new CCT form

As of version 31 and above, there are two connection routines in PSS®E – subroutines CONEC and CONET. These are in files conec.flx and conet.flx (or could be conec.for and conet.for). Historically, PSS®E users are used to generating model calls in CONEC and CCONET subroutines. The problem however in having model calls in CONEC and/or CONET is that the calls generated have hardcoded numbers which could be different from one case to another. As a consequence, there was a need to on the part of users to compile the conec and/or conet files even though they may either be using models that are supplied as part of PSS®E installation or are using models for which they have already have the model dll. In other words, there was an implicit need for a compiler for every user that had dynamic cases with model calls in conec and/or conet files.

At PSS®E version 33, the goal was to preclude the need for generating model calls in the conec and conet files. To accommodate this, several changes were introduced in the way the CONEC and CONET called models were written. In view of the changes introduced in version 33, the old CONEC and CONET type models can now be written in a form called the CCT type. The advantage in writing the old CONEC and CONET type models as CCT type model is that the CCT type models would not generate model calls in the connection routines, and as a consequence, PSS®E users, unless they are writing models, will not have to compile and link and hence will not require a FORTRAN compiler.

This document outlines the changes to be made in the model source to convert the CONEC and CONET type user models to the new CCT form.

Outline of Changes to convert to CONEC and CONET to CCT type

Fundamentally the changes to be made in the user model source are as follows:

1) The first step in the migration is to identify what CCT type model it is.

   CCT models could be one of the following types:

   a) Device type model or protection type model or Model of Type “other” – such models are attached to some equipment in power flow. The first argument for such model is the index of the device to which the model is attached. Thus, if the CCT type model is attached to a bus, then the first argument is the internal bus index, if it is attached to a 2-terminal dc line then the first argument is the internal index of the dc line and so on.

   b) Miscellaneous “Other” – such models are called Miscellaneous “other” model because these models are not necessarily attached to any specific device or equipment in power flow. Such models are “unattached” models and can be used to model anything that you can physically imagine. Being an unattached model the first argument for these model subroutines are what is called the “Model Instance”.

2) Having decided the CCT model type, the model call argument has to be changed. The model call argument instead of being M,J,K,L (as is the case for CONEC-CONET type models) will now be, (I, SLOT), where "I" is the device index (passed into the model by PSS®E) if the CCT model is attached to an equipment in power flow, or will be the model instance value if
the CCT model is of type "Other" and not attached to any equipment in power flow. In short, every CCT type model subroutine will have two arguments. The first argument is either the device index (for attached models, i.e., models attached to an equipment in power flow), or it is the model instance (for unattached models, i.e., models that are not attached to any equipment in power flow).

The second argument for the CCT type model subroutines is the allocation index (typically designated using a FORTRAN integer variable SLOT). The “SLOT” variable is used to obtain the starting CON, STATE, VAR, and ICON indices,

3) Inside the model code, starting ICON, CON, STATE and VAR indices by using the following arrays:

```
J=STRTCCT(1,SLOT) ! starting CON index
K=STRTCCT(2,SLOT) ! starting STATE index
L=STRTCCT(3,SLOT) ! starting VAR index
M=STRTCCT(4,SLOT) ! starting ICON index
```

4) The next change is in the DYDA or MODE 6 part of code (the part that writes out the DYR record of the model). The model code needs to be modified to write out the DYR record in a format compatible with the CCT model type (for details refer to the Program Operation Manual, Chapter on “Dynamic Simulation Activity Descriptions”).

5) You will then have to change the DYR data record using the format as given in the PSS/E 33 Program Operation Manual, Chapter on "Dynamic Simulation Activity Descriptions", and read the DYR file. Following this there should be no model calls in the subroutines CONEC and CONET.

Now that your CONEC and CONET called models have been converted to the CCT form, all that needs to be done is create a dll of the user model (using the Environment Manager tool that is shipped with PSS/E 33 and use this instead of using the “obj” and/or “lib” files.

Examples of converting CONEC and CONET type models to CCT form

Below we will present two examples, the first to convert a CONEC called model for performing a step change in Vref (reference voltage of AVR), and the second to convert a CONET called model to calculate the bus frequency in Hz.

Example #1: Model for performing a step change in Vref for a machine connected at bus IBUS, machine with identifier ID:

The differences between the CONEC version of the model (called SVREF), and the non-CONEC (or the CCT version) version of this model (called SVREFT) are as outlined in the Table 1 below.

<table>
<thead>
<tr>
<th></th>
<th>Model SVREF</th>
<th>Model SVREFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>Model arguments are M, J, K and L (the starting ICON, CON, STATE, and VAR indices respectively).</td>
<td>Model coded as a CCT model of type “Machine Other”. Model arguments are machine index (MC), and array allocation index (say SLOT).</td>
</tr>
<tr>
<td>2) Values of M, J, K, and L are passed into the model subroutine by the model call in CONEC subroutine.</td>
<td>Values of M, J, K, and L are obtained by knowing SLOT and using the STRTCCT arrays</td>
<td></td>
</tr>
</tbody>
</table>
### Example #2: Model for calculating the bus frequency of a given bus:

Since the bus frequency is calculated from bus voltage angle, this model can be called only after the network solution convergence. In other words, this model, is a CONET type model.

The differences between the CONET version of the model (called BFCALC), and the non-CONEC (or the CCT version) version of this model (called BFCALCT) are as outlined in the Table 2 below.

<table>
<thead>
<tr>
<th></th>
<th>Model BFCALC</th>
<th>Model BFCALCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>Model arguments are M, J, K and L (the starting ICON, CON, STATE, and VAR indices respectively).</td>
<td>Model coded as a CCT model of type “Miscellaneous Other”. Model arguments are Model Instance (MINS), and array allocation index (say SLOT).</td>
</tr>
<tr>
<td>2)</td>
<td>Values of M, J, K, and L are passed into the model subroutine by the model call in CONET subroutine.</td>
<td>Values of M, J, K, and L are obtained by knowing SLOT and using the STRTCCCT arrays</td>
</tr>
<tr>
<td>3)</td>
<td>The model dyr record would be: 0, ‘USRMDL’, 0, ‘BFCALC’, 0, 2, NI, NC, NS, NV, ICON data, CON data / Note: the IC value used above is 0</td>
<td>The model dyr record would be: MINS, ‘USRMSC’, ‘BFCALCT’, 512, 2, NI, NC, NS, NV, ICON data, CON data / Note: the IC value used above is 512</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>4)</td>
<td>The bus for which frequency calculation is to be done is part of ICON data. The bus number is part of ICON data. However, the DYRE record also requires a MINS value which is any unique number greater than 1 (see PSS®E documentation for details).</td>
<td></td>
</tr>
<tr>
<td>5)</td>
<td>Model does not require a T-entry. The model calculation logic as well as the DOCU and DYDA logic are all in the main entry (BFCACL). The DOCU and DYDA logic is in the main entry (BFCALCT), while the logic for calculating the frequency is in the T-entry (ENTRY TFCALCT).</td>
<td></td>
</tr>
<tr>
<td>6)</td>
<td>DYDA part of the code will write out the BFCALC model information in the format as shown in (3). DYDA part of the code will write out the BFCALCT model information in the format as shown in (3).</td>
<td></td>
</tr>
<tr>
<td>7)</td>
<td>Code for providing model data descriptions (CON_DSCRPT and ICON_DSCRPT) not possible. Possible to add model data descriptions in MODE 8 via CON_DSCRPT and ICON_DSCRPT.</td>
<td></td>
</tr>
<tr>
<td>8)</td>
<td>DOCU part of code requires check on IBDOCUC variable for selective DOCU purposes. Selective DOCU (e.g., DOCU of models attached to specific bus) is managed by PSS®E. No special checks are required in the model code.</td>
<td></td>
</tr>
<tr>
<td>9)</td>
<td>DOCU and DYDA part of the code has to check for F10 interrupts. DOCU and DYDA part of the code does not have to check for F10 interrupts. This is done by PSS®E.</td>
<td></td>
</tr>
</tbody>
</table>

**Summary**

Conversion of CONEC and CONET called model to a CCT type of model is fairly straightforward. The summary of changes to be made is as outlined below.

1) Identify the CCT model type, e.g., whether it is an attached model (attached to an equipment in power flow) or if it is an unattached model (model not attached to any specific equipment in power flow) in which case the model can be written as Miscellaneous Other type model.

2) Change the model call arguments - earlier the ICON, CON, STATE and VAR indices were being passed into the model, as a CCT type model the model arguments would depend on the CCT model type (for example if it is a CCT model attached to a machine, the arguments would be machine index MC, and the array allocation index SLOT, for a miscellaneous model it would be models instance MINS and SLOT, and so on).

3) Using the SLOT (array allocation index) variable and the STRTCCT array, obtain the starting indices of ICON, CON, STATE, and VAR arrays for the model.

4) Look for potential changes in the model to move any information that was earlier required as inputs in ICONs (e.g., machine bus number and machine id that was read as ICON values) to the dyre record.

5) Add model data descriptions in MODE 8 (i.e., add logic in MODE 8 to add data descriptions in CON_DSCRPT and ICON_DSCRPT).
6) The DOCU portion of the code will have to be modified to remove any checks for selective DOCU (IBDOCU checks). IBDOCU checks are required only if the model is in a CONEC and CONET form. For models not called in CONEC and CONET, selective DOCU (e.g., DOCU of models for equipments at a specific bus) is managed by PSS®E level.

7) DOCU and DYDA part of the code does not have to check for interrupts (i.e., CONINT check). For models not called in CONEC and CONET, the interrupt check is handled by PSS®E.

8) Modify the DYR record to the new form (for details on the user-written model format for CCT type models, please refer to PSSE Program Operation Manual, chapter on “Dynamic Simulation Activity Description”).
1 General

This document outlines the steps needed to convert a user-written two-terminal dc line model for older versions of PSS™E to PSS™E-31 compatible format.

Methods I and II outline two different ways to implement the conversion.

Method I should be chosen if the model must be maintained as a CONEC and CONET called user-written model (i.e., the model will continue to be called in CONEC and CONET as it was in the pre-31 version).

Method II, the PTI recommended approach, removes the model call from CONEC and CONET. In this case PSS™E calls the converted model from within PSSE (i.e., model calls will not appear in CONEC and CONET). As a consequence of the changes in the model, user would have to change the DYRE record to read the model using the 'USRDCL' keyword rather than 'USRMDL'. The DYRE record would also have to be changed appropriately (consistent with the DYRE requirements for USRDCL keyword – refer to PSS™E-31 Program Operation Manual, Volume I, chapter 5, section 5.1.1.19). This method is consistent with the way PSS™E-31 user-written dc line models are handled in PSS™E-31.

This document explains the steps for both Methods I & II. After implementing the changes, it is the user’s responsibility to ensure the correctness of the code.

2 Procedure for Implementing the Changes

Let us for instance say you have a dc line model CDCMC that has to be changed to the new format that can accept dc line names.

Assume that the arguments of the subroutine CDCMC as it exists now are I,M,J,K,L in that order, where

\[
\begin{align*}
  I & = \text{D.C. LINE NUMBER} \\
  J & = \text{STARTING 'CON' INDEX} \\
  K & = \text{STARTING 'STATE' INDEX} \\
  L & = \text{STARTING 'VAR' INDEX} \\
  M & = \text{STARTING 'ICON' INDEX}
\end{align*}
\]
2.1 Method I

There are two different ways that Method I can be implemented. The decision on which to select would depend on whether you adopt option (a) or (b) below.

a. Use the conec and conet files from Pre-31 version as in PSS™E-31
b. Execute activity DYRE and generate version 31 conec.flx and conet.flx

The changes in the model code CDCMC would depend on which option you choose.

2.1.1 Changes for Option (a)

The changes required for this option are:

1. Convert the dc line number "I" (which is in the integer format) to name format
2. Obtain the dc index corresponding to this dc line name

The steps involved are as follows:

a. Change the argument "I" in the CDCMC subroutine, and in the TDCMC entry point (T-entry, if present) to some other integer variable (say “IDCARG”). The SUBROUTINE statement would then look as follows:

```plaintext
SUBROUTINE CDCMC (IDCARG, M, J, K, L)
ENTRY TDCMC (IDCARG, M, J, K, L)
```

b. Add the following declarations in the subroutine CDCMC:

```plaintext
INTEGER IDCARG
EXTERNAL GET_2TDCINDX
```

c. In the CDCMC model code call the following statements:

```plaintext
CALL GET_2TDCINDX('CDCMC',IDCARG,I)
IF (I .EQ. 0) RETURN
```

Please note that the above statements should be placed in CDCMC and also in TDCMC (the "T-entry" point, if any), and should be the very first executable statement.

Note that GET_2TDCINDX is a PSS™E supplied function.

The rest of the code (since it already uses the variable "I") would remain as it is.
2.1.2 Changes for Option (b)

In this option, following PSS™E activity DYRE, the model call as it appears in CONEC and CONET would look like the following example:

```
CALL CDCMC ('10', 78, 876, 7, 93) * in CONEC
CALL TDCMC ('10', 78, 876, 7, 93) * in CONET
```

Note that in the above, the first argument that appears in quotes is the dc line name.

The changes required for this option are:

1. Obtain the dc index corresponding to the dc line name
2. Change the DYDA statement (MODE=6) in the CDCMC model to output the dc line name (in quotes) instead of the dc line number

The various steps involved are as follows:

a. Change the argument "I" in the CDCMC subroutine, and in the TDCMC entry point (T-entry, if present) to some other character variable (say "DEVNAME").

   The SUBROUTINE statement would then look as follows:

   ```
   SUBROUTINE CDCMC (DEVNAME, M, J, K, L)
   ENTRY TDCMC (DEVNAME, M, J, K, L)
   ```

b. Add the following declarations in the subroutine CDCMC:

   ```
   CHARACTER DEVNAME*12
   EXTERNAL FND2DC
   ```

c. In the CDCMC model code call the following statements:

   ```
   CALL FND2DC(DEVNAME,I)
   IF (I .EQ. 0) RETURN
   ```

   Note that the above statements should be placed in CDCMC and also in TDCMC (the "T-entry" point, if any), and should be the first executable statement.

   Note that FND2DC is supplied with PSS™E supplied function.

d. Change the DYDA statement to output DEVNAME instead of outputting the variable "I".
The DYDA logic and its associated FORMAT statement as it exists now would look like the following example:

```
C=========
  IF (MODE .EQ. 6) THEN
    WRITE(IPRT,507) I,(ICON(MM), MM=M, M+5), (CON(JJ), JJ=J, J+42)
  507 FORMAT(I5,' ''USRMDL'' 0 ''CDCMC'' 7 1 6 43 6 43',/
                6X,6I6,8(/5X,5G13.5)/5X,3G13.5,'/)
    RETURN
  ENDIF
C=========
```

This may be changed to:

```
C=========
  IF (MODE .EQ. 6) THEN
    WRITE(IPRT,507) DEVNAME, (ICON(MM), MM=M, M+5), (CON(JJ), JJ=J, J+4)
  507 FORMAT(' ',A,' ','USRMDL' 0 'CDCMC' 7 1 6 43 6 43',/
                6X,6I6,8(/5X,5G13.5)/5X,3G13.5,'/)
    RETURN
  ENDIF
C END MODE = 6
C=========
```

Changes may have to be made in the DOCU statement as well.

The rest of the code (since it already uses the variable "I") would remain as it is.
2.2 Method II

The steps involved in this method are as follows:

a. In this method the subroutine (CDCMC & TDCMC) arguments have to changed as follows:

```fortran
SUBROUTINE CDCMC (I, SLOT)
ENTRY TDCMC (I, SLOT)
```

In the subroutine CDCMC, declare SLOT as an integer as follows:

```fortran
INTEGER SLOT
```

The starting CON, STATE, VAR and ICON indices will then have to be derived as follows:

```fortran
J = STRT2D(1,SLOT)
K = STRT2D(2,SLOT)
L = STRT2D(3,SLOT)
M = STRT2D(4,SLOT)
```

Note that the variables, J, K, L & M are probably already declared as integers in the subroutine CDCMC.

If the CDCMC has the include of COMON4.INS (which is the recommended approach), then the STRT2D definition is already available inside the model code. If, instead of COMON4.INS, the model code has USE statements, then you would have to add the following statement:

```fortran
USE DYNAMICS, ONLY: STRT2D
```

b. Change the DYDA statement to output the dc line name instead of outputting the variable "I".

The DYDA logic and its associated FORMAT statement as it exists now would look like the following example:

```fortran
C==========
IF (MODE .EQ. 6) THEN
    WRITE(IPRT,507) I,(ICON(MM),MM=M,M+5),(CON(JJ),JJ=J,J+42)
507 FORMAT(I5,' ''USRMDL'' 0 ''CDCMC''  7 1 6 43 6 43',/,
        & 6X,6I6,8(/5X,5G13.5)/5X,3G13.5,')
    RETURN
ENDIF
C END MODE = 6
C==========
```
This may be changed to:

```fortran
C=========
IF (MODE .EQ. 6) THEN
   WRITE(IPRT,507) TRIM(DC2NAM(I)), (ICON(MM), MM=M, M+5), (CON(JJ), JJ=J, J+42)
507    FORMAT(' ''',A,'''',' ''USRDCL'' 18  1  6  43  6  43',/,
       & 6X,6I6,8(/5X,5G13.5)/5X,3G13.5,'/')
   RETURN
ENDIF
C END MODE = 6
C=========

Note that, if the model has the include of COMON4.INS, then the array DC2NAM definition
is already available inside the model code. If, instead of using the COMON4.INS, the model
code has USE statements, then following statement would have to be added:

USE PSSCM4, ONLY: DC2NAM

'TRIM' is a FORTRAN intrinsic function. This could be indicated by adding the following
declaration statement:

INTRINSIC TRIM

Changes may have to be made in the DOCU statement as well.

The rest of the code (since it already uses the variable "I") would remain as it is.