Fly! II Subsystem Walkthrough

TRI Flyhawk Electrical System

Version: 1.0

Author: Chris Wallace c_wallace@sympatico.ca

Introduction

This document provides a walkthrough of the electrical power generation subsystems for the TRI Flyhawk aircraft. All of the subsystems described here are contained in the electrical system file FLYHAWK.AMP.

Complete documentation of all subsystem object types is underway as a long-term project. However the usefulness of the 'dpnd' (Dependency) subsystem in constructing complex electrical system logic cannot easily be understood by a simple breakdown of the syntax in the .AMP file.

This document attempts to shed a little more light on how dependencies are actually used in the construction of complex system behaviour, using the power generation and distribution system of the TRI Flyhawk as an example.

dpnd Dependency Subsystem

Class: CDependency

Description:
The Dependent Subsystem is simply a functionality layer of a subsystem.

“The layer seems to have been added to confuse the documentation of Fly Subsystems” –- Paul “Simguy” Russell

The critical feature of dependencies is that they can be linked to other dependencies in order to create a hierarchical tree of functionality that impements the complex interworkings of aircraft electrical and pitot-static systems.

The <eBus> tag links this object to a ‘dpnd’ subsystem with hardware type of BUS.
The <fuse> tag links this object to a ‘dpnd’ subsystem with hardware type of FUSE.
The <swch> tag links this object to a ‘dpnd’ subsystem with hardware type of SWITCH.
The <lite> tag links this object to a ‘dpnd’ subsystem with hardware type of LIGHT.
The <annr> tag links this object to a ‘dpnd’ subsystem with hardware ID of ANNOUNCER.
The <cntr> tag links this object to a ‘dpnd’ subsystem with hardware ID of CONTACTOR.
The <flsh> tag links this object to a ‘dpnd’ subsystem with hardware type of FLASHER.
The <batt> tag links this dependency to a ‘batt’ (Battery) subsystem.
The <nvrt> tag links this dependency to a ‘nvrt’ (Inverter) subsystem.
The <genr> tag links this dependency to a ‘genr’ (Generator) subsystem.
The <altr> tag links this dependency to a ‘natr’ (Alternator) subsystem.

Several of these tags of various types can be specified as dependencies of a single object. The <_AND> and <_OR_> tags determine how multiple dependencies are combined. If the <_AND> tag is specified then all dependencies must be active in order for the object to be active. If the <_OR_> tag is specified then only one of the dependent subsystems needs to be active.

The <pxy1> and <pxy0> specify messages that are sent to other subsystems when the state of this subsystem transitions from Off to On (<pxy1> tag) or from On to Off (<pxy0> tag). These parameters can be specified multiple times.

Read Tags

Signature	Description	Type	Format
volt	Electrical Voltage	REAL	Volts
mVlt	Maximum Voltage	REAL	Volts
load	Electrical Load	REAL	Amps

freQ	AC Electrical Circuit Frequency	REAL	Hz
eBus	Logical Connection to specified Bus	ID_TYPE	Dependent unique ID
fuse	Logical Connection to specified Fuse	ID_TYPE	Dependent unique ID
swch	Logical Connection to specified Switch	ID_TYPE	Dependent unique ID
lite	Logical Connection to specified Light	ID_TYPE	Dependent unique ID
annr	Logical Connection to specified Annunciator	ID_TYPE	Dependent unique ID
bool	Logical Connection to specified Dependent	ID_TYPE	Dependent unique ID
cntr	Logical Connection to specified Contactor	ID_TYPE	Dependent unique ID
flsh	Logical Connection to specified Flasher	ID_TYPE	Dependent unique ID
batt	Logical Connection to specified Annunciator	ID_TYPE	Dependent unique ID
genr	Logical Connection to specified Generator	ID_TYPE	Dependent unique ID
altr	Logical Connection to specified Alternator	ID_TYPE	Dependent unique ID
nvrt	Logical Connection to specified Inverter	ID_TYPE	Dependent unique ID
st8t	Controls the state of the dependent (On/Off)	BOOLEAN	0 (FALSE) = OFF 1 (TRUE) = ON
_AND	Logical Circuit connections are ANDed together. All circuit connections at this node must be active to create an active circuit at this node	NO_VALUE	
_OR _	Logical Circuit connections are ORed together. Just one circuit connection needs to be active to create an active circuit at this node	NO_VALUE	
hwId	Hardware ID	CODE	Do not use. Use ‘unId’ instead
pxy1	Rising Edge Proxy Msg	OBJECT	Message
rise	Value for Rising Edge Proxy Msg	REAL	Value
pxy0	Falling Edge Proxy Msg	OBJECT	Message
fall	Value for Falling Edge Proxy Msg	REAL	Value
mPol	Polling Message	OBJECT	Message
mAll	Create all three proxy messages	OBJECT	Message
tPol	Polling Method	CODE	Persistent Proxy
offV	Indicator value when node is Off	REAL	Indicator value

Set Tags

The following subsystem values can be set by sending a SETDATA message to the subsystem with the datatag set appropriately. This allows gauges, other subsystems, custom DLLs, etc. to actively modify the operational state of any subsystem.

The <st8t> tag controls the activity (On/Off) state of the dependent. The <indn> tag overrides the current indication value for the subsystem. Either the <timK> or <ratK> can optionally be used to filter the indication value of the dependent. The <timK> sets the time constant of first-order exponential filter. This parameter is used to smooth digital transitions in the indication value. The <ratK> tag causes a linear filter to be applied to the indication, with linear rate constant as specified in the tag.

Datatag	Description	Type	Format
st8t	Controls the state of the dependent (On/Off)	BOOLEAN	0 (FALSE) = OFF 1 (TRUE) = ON
indn	Indication value	REAL	Value
timK	Time Constant	REAL	Dimensionless
ratK	Rate Constant	REAL	Dimensionless
volt	Electrical Voltage	REAL	Volts
mVlt	Maximum Voltage	REAL	Volts
load	Electrical Load	REAL	Amps
FreQ	AC Electrical Freq	REAL	Hz

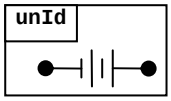
Get Tags

The following subsystem values can be retrieved by sending a GETDATA message to the subsystem with the datatag set appropriately. This allows gauges, other subsystems, custom DLLs, etc. to actively query the operational state of any subsystem.

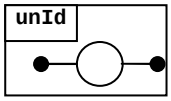
Datatag	Description	Type	Format
st8t	Controls the state of the dependent (On/Off)	BOOLEAN	0 (FALSE) = OFF 1 (TRUE) = ON
actv	Reports the circuit status	BOOLEAN	On/Off
indn	Indication value	REAL	Value
timK	Time Constant	REAL	Dimensionless
ratK	Rate Constant	REAL	Dimensionless
volt	Electrical Voltage	REAL	Volts
mVlt	Maximum Voltage	REAL	Volts
load	Electrical Load	REAL	Amps
FreQ	AC Electrical Freq	REAL	Hz

Notation

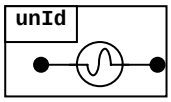
It is useful to draw out electrical system dependencies symbolically before writing the somewhat cryptic AMP file programming. I suggest the use of the following symbols for common power subsystems:



'batt' Battery

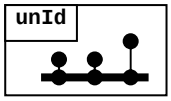


'genr' Generator

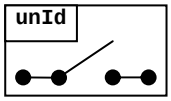


'natr' Alternator

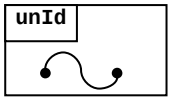
And the following for the common dependency hardware types:



'dpnd' BUS Dependency



'dpnd' SWITCH Dependency



'dpnd' FUSE Dependency

Flyhawk Electrical System Overview

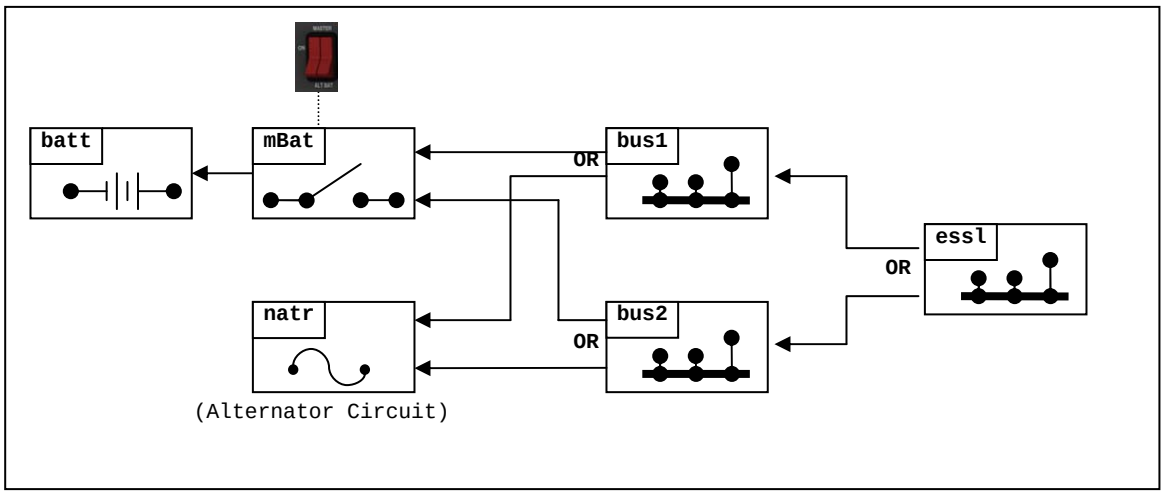
This example illustrates how the TRI Flyhawk electrical system is constructed using various types of ‘dpnd’ (Dependency) subsystems along with power generation systems such as the ‘batt’ (Battery) and ‘genr’ (DC Generator) subsystems. It traces the dependency connections working forwards from the power generation subsystems ‘batt’ (Battery) and ‘natr’ (Alternator) through several layers of ‘dpnd’ (Dependency) subsystems of various types : switches, fuses and busses. These subsystem specs are all taken from the Flyhawk systems file FLYHAWK.AMP although the individual examples may be somewhat scattered through the file.

When analyzing the electrical system of an existing aircraft, it is sometimes easier to start at the end-systems such as the radios and other avionics, and work backwards along the dependencies through the busses, fuses and switches to the power sources. However when creating a new aircraft it would typically be better to work the other way, starting with power sources and then fanning them out through switches, fuses and busses to their final destinations, the avionics systems. Detailed maintenance and/or operating documentation for the aircraft being modelled is generally required in order to accurately simulate its operation.

For this example, I have (arbitrarily) separated the TRI Flyhawk power generation and bus subsystems into three parts (although there are interconnections between the parts) : the Main Bus Circuit, the Alternator Circuit and the Avionics Bus Circuit.

Main Bus Circuit

The purpose of the Main Bus Circuit is to drive power to the main system busses ‘bus1’ and ‘bus2’, and to the “essentials” bus ‘essl’. At the “top” of the electrical chain is the battery, which is fed through the battery master switch to the two main busses, which then drive the essentials bus.



The main busses are also driven from the output of the Alternator Circuit (see below).

The battery is defined in a ‘batt’ type subsystem:

```
<subs> - Subsystem entry -
batt
<bgno> - Battery -
       <life> - amp-hour rating -
       30
       <volt> - max voltage -
       24.0
<endo>
```

Specific data tags for this battery set its voltage at 24.0 volts, and its total charge capacity at 30 amp-hours. If realistic battery drain is enabled in the simulation options, then placing a continuous load on the battery will cause it to drain over time. Note also that the battery has no dependencies associated with it; it is therefore considered to always be “On”.

The switch dependencies for the battery and alternator master switches both have a hardware type of SWITCH, and are linked to switch objects on the panel. See the Gauge Reference Guide for details on gauge types like ‘swit’ (Simple Switch), ‘knob’ (Knob) etc. for additional information. The gauge instances include a reference to the associated subsystem via the <mesg> (or other) tag in the gauge specification. This reference provides an internal “connection” between the gauge object and the subsystem object. When the switch is toggled by the user, a message is automatically sent to the subsystem setting the ‘st8t’ datatag to 0 (Off) or 1 (On).

For the master battery and alternator switches, this table shows the association between the SWITCH dependency object specified in the AMP file and the gauge object specified in the PNL file. Note that for the TRI Flyhawk, a specialized switch gauge type ‘batt’ was implemented to control both the ‘mBat’ and ‘mAlt’ switches; on other aircraft, these would typically be separately controlled by independent ‘swit’ gauge types.

AMP File	PNL File

<subs> - Subsystem entry -
dpnd
<bgno> - Master Battery Switch -
  <unId> - Unit Id -
  mBat
  <hwId> - hardware type -
  SWITCH
  <batt> -- Battery Circuit --
  batt
<endo>
	<gage> ==== GAUGE ENTRY ====
	batt
	<bgno> ==== BEGIN GAUGE ENTRY ====
	  <unid> ---- unique id ----
	  batt
	  <mesg>
	  <bgno>
	  <conn>
	  mBat
	  <dtag>

```
<subs> - Subsystem entry -
dpnd
<bgno> - Master Alternator Switch -
       <unId> - Unit Id -
       mAlt
       <hwId> - hardware type -
       SWITCH
       <fuse> - independent circuit -
       altf
<endo>
```

```
                st8t
       <endo>
       <altt> ---- alt data tag ----
       mAlt
       <batt> ---- battery data tag ----
       mBat
       <size> ---- switch size ----
       109
       576
       36
       50
       <bmap> ---- bitmap name ----
       skyhpowr.pbg
       <csru> ---- up cursor ----
       mpflipup.csr
       <csrd> ---- down cursor ----
       mpflipdn.csr
       <sfxu> ---- up sfx ----
       switchup.wav
       <sfxd> ---- down sfx ----
       switchdn.wav
       <help> ---- popup help ----
       Alternator/Battery Switch
<endo> ==== END GAUGE ENTRY ====
```

Note that the master alternator has a subsystem dependency, to the ‘altf’ FUSE. This fuse is described later in the document.

Next are the definitions for the Main busses. Main bus 1 ‘bus1’ and Main Bus 2 ‘bus2’ have identical dependencies on the alternator fuse and main battery switch subsystems.

```
       <subs> - Subsystem entry -
       dpnd
       <bgno> - Bus #1 -
              <unId> - Unit Id -
              bus1
              <hwId> - hardware type -
              BUS
              <fuse> - independent circuit 1 -
              natr
              <swch> - independent circuit 2 -
              mBat
       <endo>
       <subs> - Subsystem entry -
       dpnd
       <bgno> - Bus #2 -
              <unId> - Unit Id -
              bus2
              <hwId> - hardware type -
              BUS
              <fuse> - independent circuit 1 -
              natr
              <swch> - independent circuit 2 -
              mBat
       <endo>
```

Note that there are two dependencies specified, but no <_AND> or <_OR_> tag present. The <_OR_> tag is the default, so these busses will both be Active when either the alternator is On, and/or the battery master switch is On.

The two <eBus> tags indicate that this bus is Active whenever either ‘bus1’ or ‘bus2’ are active. (Actually, since we noted above that ‘bus1’ and ‘bus2’ have identical dependencies, there is not really any need to have two dependencies here).

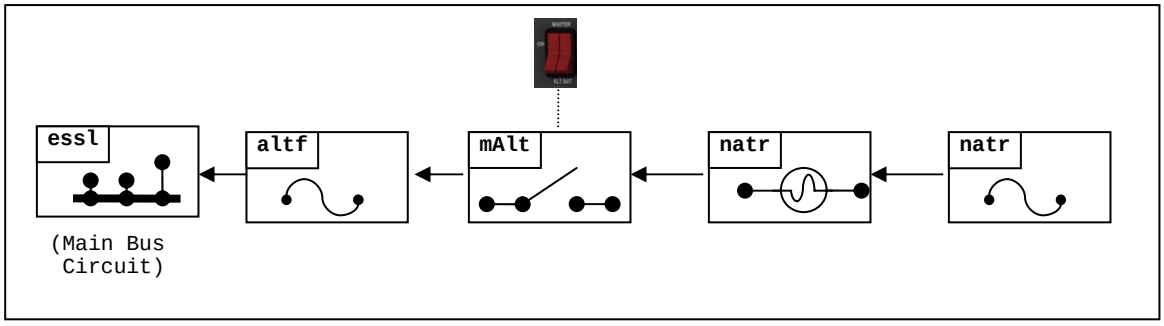
The essential bus drives the critical electrical components such as the annunciators and starter circuits. Its dependency definition is:

```
<subs> - Subsystem entry -
dpnd
<bgno> - Essential Crossfeed Bus -
       <unId> - Unit Id -
       essl
       <hwId> - hardware type -
       BUS
       <eBus> - independent circuit -
       bus1
       <eBus> - independent circuit -
       bus2
<endo>
```

The essentials bus is used directly by several subsystems such as the annunciator lights, starter/igniter system, etc. It is also a dependency of other electrical subsystems in the Alternator and Avionics Bus Circuits.

Alternator Circuit

The Alternator Circuit contains all of the logic surrounding the engine-driven alternator power source. The output of the Alternator Circuit is the ‘natr’ fuse which drives power to the Main Bus Circuit, and to any other subsystems which need to directly monitor the output of the circuit. At first glance it may appear that this is a “short circuit” since the essentials bus is a dependency of the Alternator Circuit. However this is not the case; all it means is that the essentials bus must first be powered up with the battery, after which (and after other alternator dependencies are satisfied such as the engine running, master alternator switch on, etc.) the Alternator Circuit can “take over” driving the main busses.



The first dependency in this circuit is the alternator field fuse, which isolates the Alternator Circuit from the Main Bus Circuit. The entire alternator circuit can be independently failed simply by failing the ‘altf’ fuse.

```
<subs> - Subsystem entry -
dpnd
<bgno> - Alternator Field Fuse -
       <unId> - Unit Id -
       altf
       <hwId> - hardware type -
       FUSE
       <eBus> - independent circuit -
       essl
```

```
<endo>
```

The master alternator switch is dependent both upon the panel gauge switch (see above) and on the alternator field fuse.

Next in line is the alternator itself, defined in a ‘natr’ type subsystem:

```
<subs> - Subsystem entry -
natr
<bgno> - Alternator -
       <swch> - circuit -
       mAlt
       <mxld> - max capacity (amps)
       60
       <loRg> - least RPM regulatable -
       550
       <volt> - max Voltage -
       28.0
<endo>
```

This alternator is rated to provide 28.0 volts at a maximum of 60 amps continuous current. All alternators are engine-driven, and the <loRg> tag indicates the minimum engine RPM at which the alternator can function. This would normally be a couple of hundred RPM less than the idle RPM for the engine.

The alternator has a single dependency on the master alternator switch ‘mAlt’. As described above, this SWITCH dependency is controlled solely by the alternator switch on the main panel. When this switch is in the On position, and the engine is running, then the alternator subsystem will be delivering power.

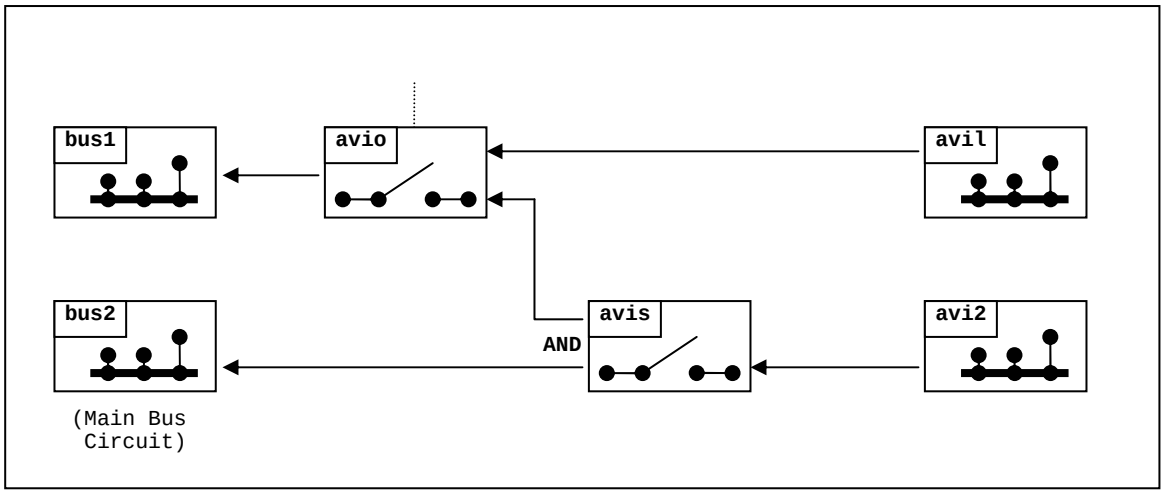
Finally the output of the alternator is fed into the alternator fuse:

```
<subs> - Subsystem entry -
dpnd
<bgno> - Alternator Fuse -
       <unId> - Unit Id -
       natr
       <hwId> - hardware type -
       FUSE
       <altr> -- Alternator Circuit --
       natr
<endo>
```

The <altr> tag creates a dependency between the FUSE and the ‘natr’ subsystem described above. Whenever the alternator is On and delivering power, this FUSE will be Active.

Avionics Bus Circuit

The purpose of the Avionics Bus Circuit is to drive two main avionics busses, ‘avi1’ and ‘avi2’. These busses are used as dependencies to most of the radio and other avionics systems in the aircraft.



Two avionics fuses are defined which simply mirror the state of the associated main busses from the Main Bus Circuit:

```
<subs> - Subsystem entry -
dpnd
<bgno> - Circuit Breaker -
       <unId> - unit id -
       avi1
       <hwId> - hardware type -
       FUSE
       <eBus>
       bus1
<endo>
<subs> - Subsystem entry -
dpnd
<bgno> - Circuit Breaker -
       <unId> - unit id -
       avi2
       <hwId> - hardware type -
       FUSE
       <eBus>
       bus2
<endo>
```

These avionics fuses provide power to the master and secondary avionics switches. The master avionics power is controlled by the avionics master switch on the panel:

AMP File	PNL File

<subs> - Subsystem entry -
dpnd
<bgno> - Avionics Switch (Master) -
  <unId> - Unit Id -
  avio
  <hwId> - hardware type -
  SWITCH
  <fuse> - independent circuit -
  avi1
<endo>
	<gage> ==== GAUGE ENTRY ====
	swit
	<bgno> ==== BEGIN GAUGE ENTRY ====
	  <unid> ---- unique id ----
	  avio
	  <mesg> ---- Subsystem Message ----
	  <bgno>
	  <conn> ---- data tag ----
	  avio
	  <user> ---- user data ----
	  HARDWARE,SWITCH
	  <endo>
	  <size> ---- x,y,xsize,ysize ----
	  343
	  576
	  33
	  49
	  <bmap> ---- bitmap name ----

```
        skyhavpw.pbg
        <csru> ---- up cursor ----
        mpflipup.csr
        <csrd> ---- down cursor ----
        mpflipdn.csr
        <sfxu> ---- up sfx ----
        switchup.wav
        <sfxd> ---- down sfx ----
        switchdn.wav
        <help> ---- popup help ----
        Avionics Power
<endo> ==== END GAUGE ENTRY ====
```

The secondary master switch has a hardware type of SWITCH, however it is not connected to a switch on the panel; rather it is dependent only upon other subsystems:

```
<subs> - Subsystem entry -
dpnd
<bgno> - Avionics Switch (Slave) -
       <unId> - Unit Id -
       avis
       <hwId> - hardware type -
       SWITCH
       <fuse> - independent circuit -
       avi2
       <swch> - independent circuit -
       avio
       <st8t> - hardware state -
       1
       <_AND> -- AND Independent Circuit States --
<endo>
```

So in essence, both avionics switches ‘avio’ and ‘avis’ are controlled by the master avionics switch on the panel; the ‘avio’ switch directly through the message connection to the ‘swit’ gauge type, and the ‘avis’ switch indirectly through the ‘avio’ switch.

Finally, the avionics busses ‘avi1’ and ‘avi2’ are defined. If you examine avionics and other electrical subsystems elsewhere in the AMP file, you will notice that they practically all have <eBus> dependencies back to one of these avionics busses.

Avionics bus 1 ‘avi1’ is dependent only on the ‘avio’ switch described above, while avionics bus 2 is dependent only on the ‘avis’ switch.

```
<subs> - Subsystem entry -
dpnd
<bgno> - Avionics Bus #1 -
       <unId> - Unit Id -
       avi1
       <hwId> - hardware type -
       BUS
       <swch> - independent circuit -
       avio
<endo>
<subs> - Subsystem entry -
dpnd
<bgno> - Avionics Bus #2 -
       <unId> - Unit Id -
       avi2
       <hwId> - hardware type -
       BUS
       <swch> - independent circuit -
       avis
```

<endo>

Conclusion

Hopefully this walkthrough of the power generation and bus systems for a relatively simple aircraft will help to demystify the inner workings of Fly! II electrical systems. The aircraft designer has complete freedom to implement as much or as little complexity in the electrical systems, depending on how accurately one wishes to reproduce the behaviour of the real-life aircraft. The ‘dpnd’ (Dependency) subsystem provides almost limitless flexibility for implementing complex, realistic system behaviour.