Macro.

This capability of the CRHM program allows users to create simple modules suitable for testing algorithms and for diagnosing CRHM model output.

Local Variables.

Local variables are defined using the the keyword "var". For example "var i", "var i var j" or "var i, j".

CRHM variables.

CRHM variables as those defined in the "declreadobs", "declgetvar", "declparam", "declvar" and "declobs" declarations. Note that the latter three types are defined in the current macro module and the first two types are derived from other CRHM modules in the model. The macro commands are enclosed in a for loop which is executed NHRU times. A local variable "hh" is defined so that values for every iteration may be saved in the CRHM macro module variable output. Note that local variables are not accessible outside the macro module except by saving their values into CRHM variables.

Arithmetical Operators.

1. +, -   addition/subtraction
2. *. /   multiplication/division
3. ^   exponentiation
4. %   modulus
5. (...)   brackets enclosing an arithmetical expression.
6. [n]   array element index. Order for 2-D is [hh][ll], i.e. hru first. Elements are referenced 1, 2, 3, 4 ... Cannot be an expression. Use var i; i = J+k; array[i], not array[j+k].

Logical Operators.

1.	||	OR.
2.	&&	AND.
3.	!=	Not equal.
4.	==	Equal.
5.	<=	Less Than or Equal.
6.	<	Less Than.
7.	>=	Greater Than or Equal.
8.	>	Greater Than.
9.	!	Logical Not. (Faulty)

Control Statements.

if (condition) ... else ... endif

- multiple statements or none are permitted in the TRUE and FALSE fields.
- The "if" statement must always be followed by a closing "endif" statement.
- "else" is optional if there are no FALSE statements to execute.
- Multiple "if" statements are permitted.
- "if" statements can appear within other "if" statements.
- Lowercase must be used for "if', "else" and "endif".
- "else" "if" must always be entered as two separate words.
- Example :- if ... else if ...endif endif.

while(condition) ... endwhile.

- while condition is true the code in the body of the while is executed.

for(initialization; condition; increment) ... endfor.

- no field may be left empty.
- initialization sets initial value of optional loop counter.
- condition when FALSE terminates the loop.
- condition can be a compound logical statement, e.g. "for (X = 0; lastX - X > 0.01 && max < 1000; max = max +1)".
- initialization and increment fields can have multiple statements separated by commas, e.g. "for (i = 0, j = 0; i < 10; i = i+1, j = j+2)".

Subroutine Library.

1. sin deg, where deg(°)
2. cos deg, where deg (°)
3. exp
4. log
5. log10
6. min
7. max
8. estar t
9. patmos Ht, in kPa, where Ht (m) is the height.
10. rhoa t, ea, Pa, in (kg/m^3), where t (°C), ea (kPa) and Pa (kPa) is the atmospheric pressure.
11. spec_humid ea, Pa, in (kg/kg) where ea (kPa) and Pa (kPa) is the atmospheric pressure.

12. PI

13. DAY - current day

14. MONTH - current month

15. YEAR - current year

16. JULIAN - Julian day of the year.

17. FREQ - number of time intervals in a day.

18. STEP - current interval starting at 1.

19. GROUP - Current group index. 1 to maximum number of groups.

20. STRUCT - Current struct index. 1 to maximum number of structs.

21. FIRSTINT - True for the first interval of the day. When STEP % FREQ equals 1.

22. LASTINT - True for the last interval of the day. When STEP % FREQ equals 0.

23. NO_DISPLAY - If variable is set to this value it will not display.   When exported creates a sparse file.

24. RAND - random numbers between 0.0 and 1.0.

25. ReadAheadObs - write to this function to read observations before and after the current interval. Writing -2 will cause all observations referenced by a "declreadobs" declaration in this module, to refer to the interval two periods earlier, +2 to the period two intervals later and 0 will return module read observations to the current interval. Reading from ReadAheadObs returns the status, 1 - error (outside available observation range). HRU_OBS is not used to access the observation. Observations are read in sequence as stored in the file.

26. WriteAheadObs - use this function to write the values of the current interval observations to permanent storage. Useage is to read from the desired

interval using ReadAheadObs function. Then changing the value of the desired observation and then writing the new values to observation storage using the function WriteAheadObs with the same interval offset..

Macro Declarations.

N.B spaces may be included in text fields if the entire field is enclosed in double quotes.

To create a parameter in the module,

declparam, param, NHRU, 0.2, 0.0, 1.0, "my description", "(my units)"[,Int].

Parameter macro variables are by default floating point. If is necessary to use an existing CRHM integer parameter this can be done by adding "Int" to the end of the normall call;

declparam, inhibit_evap, NHRU, [0], 0, 1, "inhibit evapatation, 1 -> inhibit", "()", "Int".

To manage 2-D parameters the dimension NDEFN is implemented.

declparam, Distrib, NDEFN, [1.0], 0.0, 100.0, "Test 2D parametert", "()"

The order of element access to Distrib [HRU][LAY].

To change the value of a CRHM parameter declared in another module,

declputparam, module_name, variable_name, (units).

To use a CRHM observation within the module,

declreadobs, t, NOBS, description, (units). N.B. access is limited to the available observations. Last available observation is used to satisfy any remaining requests.

To use a CRHM observation function within the module,

declobsfunc, t, tfunc, FUNC. N.B. access is limited to primitive observations. FUNC from "AVG, MIN, MAX, DTOT, POS, TOT, FIRST, LAST, MJ_W and

W_MJ".

To create a new CRHM variable for the module,

declvar, OutVar, NHRU, description, (units) [,Int].

decldiag, OutVar, NHRU, description, (units)[,Int].

decllocal, OutVar, NHRU, description, (units)[,Int].

To manage 2-D parameters the dimension NDEFN is implemented.

declvar, Test_NDEFN, NDEFN, "Test 2D variable", ().

The order of element access to Test_NDEFN [HRU][LAY].

To create a new **state** CRHM variable for the module,

declstatvar, OutVar, NHRU, description, (units).

To create a new CRHM local variable for the module. N.B. this a variable local to this module. Not to be confused with a parser local variable.

decllocal, OutVar, NHRU, description, (units).

To use a CRHM variable from another module,

declgetvar, module_name, variable_name, (units).

To use a CRHM variable declared in another module and alter its value,

declputvar, module_name, variable_name, (units).

To use a CRHM parameter from another module,

declgetparam, module_name, variable_name, (units).

To use a CRHM parameter declared in another module and alter its value,

declputparam, module_name, variable_name, (units).

To create a CRHM observation from existing observations, parameters and variables,

declobs, t2, NHRU, description, (units). N.B. if observation is already defined by an observation file - does nothing.

To force modules into a desired loading order,

setpeer, PeerVar, PeerRank, where PeerVar is a CRHM variable that the current module must be loaded after and the PeerRank is the offset at this level.

This command is required when a module has no input variables to allow CRHM to determine the position of the module in the model order. A typical case is a module whose inputs consist of observations. Automatically it will be loaded early in the model even if it uses declared observations from other modules because all types of observations have the same priority.

To force the module to load after a declared observation has been calculated set PeerVar to 'ObsName#'. The # symbol differentiates between a variable named 'ObsName' and a declared observation named 'ObsName'.

The PeerVar can be a variable that is accessed using a declputvar by the module. However, the module will be ranked to the module originally declaring the variable. Examples of these variables are "SWE", Sd, soil_moist, soil_rechr, hru_actet and hru_cum_actet.

Macro Structure.

1. The first line of a Macro is its name. This is the name that it is identified by in the model. Macro and Module names must be unique. Text after the module name is handled as the module description.

2. Next follows the declaration section. Each declaration is on a new line.

3. The "command" line ends the declaration section and begins the code to be executed.

4. The execution code is free format and may be indented and commented.

5. The end of the macro definition is indicated by the "end" statement on a

new line.

Comments.

Code may be documented line using "//". Any text after the "//" is ignored and handled as a comment.

To use spaces in declaration descriptive(text) fields enclose the desired text in double quotes, e.g. declparam, param, NHRU, 0.2, 0.0, 1.0, "my description", "(my units)"

Array references are in the range of 1 to the maximum number of HRUs.

Element[0] is illegal. When using a standard observation variable the element access is [1], e.g. T[1], u[1] etc. If the array element is not specified it will default to [1]. Not recommended.

When accessing observations, the element is limited to the maximum defined element for the observation.

Macro Edit Screen.

This screen is a simple text editor. At present no "smarts" are built in. The screen has the capability to cut and paste to and from itself and to and from other applications. Macro modules can be saved from the screen using the File menu. The default file extension is "*mcr". These macro files are never used by CRHM and are for the use of the user only.   The two buttons allow the user to save the screen changes to CRHM or cancel current changes and return to the last saved CRHM screen in the model.

To create a new line use CTRL + Enter.

When loading a macro file (*mcr), it will by default insert the text into the edit screen at the position of the cursor. However, if the edit screen has a selection, it will be replaced by the contents of the file.

When saving a macro, the entire edit screen is saved to the file unless there is a selection and in that case only the selected text will be saved.

Saving Macros.

Macros are automatically saved to the CRHM project file when the model is saved as a project in the main screen file menu. A macro may also be saved as a file in the Macro Edit Screen for import into another project. The file extension used is ".mcr". Since CRHM loads executable Macros from the project file, to utilise code in a "*.mcr" file the file must be loaded into CRHM using the Macro Edit Screen and then the project saved. Exit from CRHM and then re-run CRHM and load the project file.

Flow Screen.

Since macro modules used for debugging may not be required to satisfy inputs to the current CRHM model, CRHM will detect them as unused. To keep the macro modules, always select "NO" in the "Remove module" dialogue box. Macro declared observations are labelled with a trailing "#". For example the Macro declared observation "MyObs" will be displayed as "MyObs#". This notation differentiates declared Macro Observations from declared Macro Variables.

Declared Observation Linking Priority.

When a model is run and an Observation is available from a file (field observation) and also from a Macro, CRHM by default will use the observation from the file.

When a Macro defines an observation that should have a higher priority than the file observation, its name should have a trailing "#' sign, e.g. "MyObs#" which will display in the flow screen as "MyObs#". When this convention is used, "hard code" Modules have to contain extra code to handle the special name.

Macro Example.

The following macro definitions demonstrate the following features.

1. Macros are named by the user.

2. Multiple macros may be defined at once.

3. Standard CRHM parameters allow macro physical outputs to be easily set and modified like normal CRHM modules.

4. Any Observations from the CRHM model may be accessed.

5. Any CRHM module/macro output variable may be accessed.

6. CRHM variable outputs may be generated to be used by other macro or standard CRHM modules.

7. Local variable values are preserved from time interval to time interval.

8. Writer should provide a description and units for the variables and parameters used to permit CRHM to supply help information to the user.

```
MyMacro1  optional module description

declparam, param, NHRU, 0.2, 0.0, 1.0, "my description", (my_units)

declreadobs, t, NOBS, description, (units)

declvar, OutVar, NHRU, description, (units)

declvar, XOutVar, NHRU, description, (units)

declgetvar, obs, hru_t, "(°C)"

command // code to be executed

OutVar[1]=param[1]*t[1] OutVar[2]=param[2]*t[1] OutVar[3]=param[3]*t[1] //
array element access by numeric value (range 1 - # HRUs)

var i i=i+1 XOutVar= sin(i) var j j=i+180 XOutVar[2] = sin(j) XOutVar[3] =
cos(PI/36*i)

end // end of code and end of module definition

MyMacro2 // beginning of next module definition
```

```
declparam, param2, NHRU, 0.2, 0.0, 1.0, description, (units)

declreadobs, t, NOBS, description, (units)

declvar, Z, NHRU, description, (units)

declvar, Y, NHRU, description, (units)

declgetvar, Macro1, OutVar, (units)

command

Z[hh]=param2[hh]*t[1]

Y[hh] = param2[hh]*OutVar[hh]

end
```

Evaporation Example.

```
Evaporation // module name

declparam, A, NHRU, 0.023, 0.0, 1.0, "description", (mm/day) // declarations

declparam, B, NHRU,17.8, 0.0, 100.0, "description", (°C)

declparam, Zref, NHRU,1.5, 0.001, 100.0, Zref, (m)

declparam, Zwind, NHRU,10, 0.001, 100.0, Zwinf, (m)

declparam, Z0, NHRU,0.001, 0.001, 100.0, Zo, (m)

declvar, EvapAlg, NHRU, "evaporation_algorithm", (MJ/(m2/day))

declvar,cum, NHRU, "cum_evaporation_algorithm", (mm)

declvar,Ra, NHRU,Ra, (s/m)

declgetvar, obs, hru_tmean, "(°C)" // mean air temperature

declgetvar, obs, hru_tmin, "(°C)" // minimum air temperature
```

```
declgetvar, obs, hru_tmax, "(°C)" // maximum air temperature

declgetvar, obs, u, "(m/s)" // wind velocity

command // module code

var U U=max(u[0], 0.2) // assume minimum wind velocity to prevent divide by zero errors

Ra[hh] = log(Zref[hh]/Z0[hh])* log(Zwind[hh]/Z0[hh])/0.4^2*U

EvapAlg[hh] =-A[hh]*( hru_tmean[1] - B[hh])*Ra[hh]*( hru_tmax[1] - hru_tmin[1])^0.5*1/(245*2.501)

cum[hh] = cum[hh] + EvapAlg[hh]

end // end of module
```

Macro Implementation of C++ module.

To relate to a practical example we will design a macro to simulate the module ClassExample described earlier which converts interval net radiation in MJ/(m2-Int) calculated by an earlier module to mm/(m2-Int) of water, i.e. kg/(m2-Int) of water. The air temperature from an observation is required to carry out the conversion. Two other outputs are calculated as a fraction of the module output. These fractions are specified by the parameters F_Qg and F_Qs.

```
Example // name of micro module

declreadtobs(t, OBS, Temperature, (°C))

declgetvar(netall, net, "(MJ/m^2*int)")

declparam(F_Qg, NHRU, 3*0.2, 0.0, 1.0, Qg=F_Qg*Rn, ())

declparam(F_Qs, NHRU, [0.0], 0.0, 1.0, Qs=F_Qg*Rn, ())

declvar(Rn, NHRU, net, (mm/Int))

declvar(Qg, NHRU, ground_flux, (mm/Int))
```

```
   declvar(Qs, NHRU, storage_flux, (mm/Int))
```

// The algorithm code to be executed every time interval and for every NHRU is written into the command area**.** The program code follows:

```
   command // code is executed for number of HRUs with hh varying between 1 and # HRUs.

      Rn[hh] = net[hh]/(2.501-0.002361*t[1])

      Qg[hh] = Rn[hh]*F_Qg[hh]

      Qs[hh] = Rn[hh]*F_Qs[hh]

   end
```

Since the command code applies to every HRU, it is executed inside a for loop. The output variable Rn, is calculated from the observation temperature and an output variable net calculated in another module. Outputs Qg and Qs from this module are the product of the output Rn and the parameters F_Qg and F_Qs.

Example of "for" and 2-D arrays.

```
Test_declvar

declvar, Test_NDEFN, NDEFN, "Test 2D variable", ()

declvar, Test_NDEFN2, NDEFN, "Test 2D variable", ()

declparam, Test_par_NDEFN, NDEFN, [1.0], 0.0, 100.0, "Test 2D parametert", "()"

command

var Fred [NHRU][7]

var ll

ll = 1
```

```
for(ll = 1; ll <= NHRU; ll = ll +1)

   Fred[hh] [ll] = Test_par_NDEFN[hh][ll]*5

   Test_NDEFN[hh][ll] = Test_par_NDEFN[hh][ll]

   Test_NDEFN2[hh][ll] = Fred[hh][ll]

endfor

end
```

Example of accessing variables and parameters from another module, in this case pbsm_M.

```
Test_getvar

declvar, Test_NDEFN, NDEFN, "Test 2D variable", ()

declvar, Test_NDEFN_P, NDEFN, "Test 2D variable", ()

declgetvar, pbsm_M, Results, ()

declparam, distrib, NDEFN, 1.0, 0.0, 100.0, "Test 2D parametert", "()"

command

var ll

for(ll = 1; ll <= NHRU; ll = ll +1)

  Test_NDEFN[hh][ll] = Results[hh][ll]

  Test_NDEFN_P[hh][ll] = distrib[hh][ll]

endfor

end
```

As always when sharing a parameter between modules, all values of the parameter should be made the same in every module, then the project saved and reloaded when the shared parameter should appear only in the "basin" module.

Known problems.

If a major change is made to a macro, i.e. insertion or deletion of a declaration, the user should exit from the macro entry screen and immediately save the project and then exit from CRHM. When CRHM is restarted it will execute properly. At present CRHM is not handling some aspects of allocation/deallocation of variables correctly.

Groups and Sructures.

Group.   A collection of modules executed in sequence for all HRUs.

After using CRHM for a while, it was found to be inconvenient to always handle the individual modules. To overcome this groups were introduced. A group module, is a collection of modules which can be used in place of specifying the individual modules. When the group is defined the modules must be specified in the correct execution order. Use of groups and a relevant naming convention allow models to be easier implemented and understood. Unecessary detail can be hidden from the flow diagrams and documentation. The larger building blocks simplify the implementation of larger models.

Since different groups can have the same variable outputs in a model it is necessary to enhance the output variable names so that they do not conflict with one another. Variable names can already be long to be meaningful, a short suffix seemed to be the best way to differentiate the repeated names and the root name still to be recognizable. Suffix @A, @B, @C... are used where @A is used for outputs of the first group, @B is used for the next group etc.

Group application:

1. If groups are defined with the same modules, it is possible to execute the models in parallel using different parameters or driving observations.
2. If groups are defined as different models, it is possible to execute the models in parallel using identical parameters and driving observations to check different responses.

Structure. A parallel collection of modules. Only a selected one of them is executed for any HRU in a time step.

Again after using CRHM it was found that it was not always desired to execute the same module for every HRU. A structure handles this situation. The selection of the module for every HRU can be programmed statically, e.g. example 1. below or dynamically by using a preceding module to select the module to be used for the current time step, e.g. example 2. below.

Since structures can have the same variable outputs in a model it is necessary to enhance the output variable names so that they do not conflict with one another. Variable names can already be long to be meaningful, a short suffix seemed to be the best way to differentiate the repeated names and the root name still to be recognizable. Suffix @a, @b, @c... are used where @a is used for outputs of the first structure, @b is used for the next structure etc. Groups use an upper case suffix while structures use a lower case suffix.

Structure application:

1. Comparison of algorithms; it is possible to specify a different module, say from evap, evapD, ShuttleWaite and ShuttleWaiteD for every HRU and track the different responses. Some of the modules, say evap, may be used more than once with parameters selecting Granger, Priestley-Taylor and Penman-Monteith giving more combinations.
2. Sometimes HRUs require diverse modules to be representative of the unit. An example would be forested and open farmland. By using the structure capability a general model can be customised to handle individual HRUs differently.
3. Sometimes HRUs change their characteristics due to excess water or lack of it. Using a structure, the module selection can be dynamically changed. If an HRU can experience dry spells, moderate rainfall and very wet conditions with flooding then it might be desireable to treat it using a grasslands module, wetlands module or a slough module respectively. The decision about which module to use would be made by a preceding module based upon the availability of moisture.

AKA Interaction with Groups.

In the normal useage of AKA in non-group models, the variables and observations are addressed uniquely by specifying the varable or observation and the module. However, with groups, variables and observations all are referenced by the group name. This lack of resolution means that the source and destination of variables and observations cannot be defined precisely. For example, if the user attempts to change say Qsi to the declared observation Qsi#, all occurrences of Qsi would be changed even the input to the module generating Qsi#, causing a loop. To prevent this from happening, if an attempt is made to change an input of a module to the same name as one of its outputs, it will be ignored.

Declared observations, e.g. Qsi# do not have future data available to be able to generate any daily function, i.e. mean, max etc. CRHM detects these calls and leaves the observation as a simple observation, i.e. Qsi. In most situations this is the most desireable selection anyway.

Macro declgroup.

The Macro Group feature allows a number of modules to be grouped under one name and treated as one module.

The first line of the macro is the module or group name. In this case MyGroupA, MyGroupB and MyGroupC.

The following lines up to the *command* token, list the modules making up the group. They must be arranged in the correct execution order.

The tokens *command* and *end* complete the definition of the group.There should not be any lines between *command* and *end*.

Multiple macro groups and macros can be defined together.

```
MyGroupA

declgroup  // defaults to  number of HRUs defined in the model.

    obs

    calcsun

command

end

MyGroupB

declgroup 5  // five  HRUs.

    intcp

    pbsm

    albedo

    netall
    ebsm

    evap

command

end

MyGroupC

declgroup 0  // defaults to  number of HRUs defined in the model.

    crack

    smbal

    route

command

end
```

Module Naming Convention.

Group variable names cannot use the original variable name otherwise their would be a naming conflict. To avoid this problem, the first macro group defined has the suffix "@A" the next "@B", "@C" etc.Because of this the search order defined in the a following section has been adopted.

Group parameters.

The group will have all the parameters of every module in the group.   If a parameter is used by more than one module in the group these modules all share the same parameter values.

Module Linking Order.

Modules can use "*" or explicitly specify the source module name,   e.g. *.hru_t and Obs.hru_t for the linking to work correctly. Obs.hru_t can only link to the module"Obs" and no other module or group.

The module linking search order is as follows.

1. Specific module, e.g. Obs.hru_t. Will only match Obs.hru_t.
2. Wild root module or group, e.g. *.hru_t and *.hru_t@A. *.hru_t would match any module with an output hru_t. *.hru_t@A will only match the hru_t output of the first group defined i.e. with suffix @A
3. Wild group module stripped of its group suffix. *.hru_t@A is reduced to *.hru_t before searching. *.hru_t@A will match any module with an output hru_t.
4. Wild group module stripped of its group suffix and a search is made of any groups after their group suffix has been removed. *.hru_t@A is reduced to *.hru_t before searching all groups for *.hru_t, i.e. after their suffix has been removed.

CreateGroup in the Macro edit menu.

This command allows an existing project to be convertede to a group.   The new group has the name of the original project with "_A" appended.   If the project is added multiple times the other groups will have "_B", "_C", etc. appended.

Multiple different projects may be added in any order. In every case the suffix "_A", "_B" etc. will be added.

The final model is build in the usual way by going to the menu "Build/Construct" and selecting the groups and building as normal. The number of HRUs in each group is determined from the project that the group was created from originally. At this time all parameters are CRHM default values.

After the model is built the original project parameter values may be moved into the new model by proceeding to the Parameter menu and loading the parameter file - "CreateGroup.par". Care should be taken to use the current directory as the same file name is always used. If this step is not taken parameter values will default to the module parameter default values.

During the preceding steps the number of HRUs should not be changed as the number of HRUs in the original project and the generated groups must be identical or they will not updated to the values in "CreateGroup.par".

The new project should be saved at this time and re-loaded before any further editing is carried out.

The number of HRUs in any group can be changed by inserting/changing the value on the "declgroup" instruction in the Macro defining the group. Note that if the value is missing or equal to 0, the number of HRUs in the group will be the global value.

When the number of HRUs is reduced the parameters are truncated. If the number of HRUs is increased the last value is duplicated as often as necessary. An exception is a serial parameter, "1, 2, 3!" for example, then the series is expanded.

It is important the names of the groups are not changed or the link between the original parameter values saved in the file "CreateGroup.par" and the groups will be broken causing the error "Unknown Parameter in parameter file" will occur for every parameter.

Macro declstruct.

The Macro Structure feature allows for a module to be chosen from a group of modules at execution time. An example of its useage is an area which is a wetland in wet years and a grassland during dry periods. The module used to represent the area can be chosen from the structure selection by another module setting the "HRU_struct" parameter for the structure module.

The first line of the macro is the module or structure name. In this case MyStructA, MyStructB and MyStructC.

The following lines up to the *command* token, list the modules making up the structure. They can be arranged in any order and are addressed as 1, 2, etc. to n.

The tokens *command* and *end* complete the definition of the group.There should not be any lines between *command* and *end*.

Multiple macro groups/structures and macros can be defined together.

```
MyStructaA

declstruct

    MyMacro1  // executed when "MyStructaA HRU_struct" = 1

    MyMacro2

command

end

MyStructaB

declstruct

    evap

    evap_Resist
```

```
command

end

MyStructaC

declstruct

    route

    netroute

command

end
```

Module Naming Convention.

Group variable names cannot use the original variable name otherwise their would be a naming conflict. To avoid this problem, the first macro structure defined has the suffix "@a" the next "@b", "@c" etc. Because of this the search order defined in a following section has been adopted. N.B. uppercase designates a group and lowercase designates a structure.

Structure parameters.

Since a structure can contain a diverse collection of modules having different parameters the structure will have all of these parameters. However, for any HRU only the parameters used by the module selected need to be defined. Any others can be left at their default values for that HRU.

Module Linking Order.

Modules can use "*" or explicitly specify the source module name, e.g. *.hru_t and Obs.hru_t for the linking to work correctly. Obs.hru_t can only link to the module"Obs" and no other module or group.

The module linking search order is as follows.

1. Specific module, e.g. Obs.hru_t. Will only match Obs.hru_t.
2. Wild root module or group, e.g. *.hru_t and *.hru_t@A. *.hru_t would match any module with an output hru_t. *.hru_t@a will only match the hru_t output of the first group defined i.e. with suffix @a
3. Wild group module stripped of its group suffix. *.hru_t@a is reduced to *.hru_t before searching. *.hru_t@a will match any module with an output hru_t.
4. Wild group module stripped of its group suffix and a search is made of any groups after their group suffix has been removed. *.hru_t@a is reduced to *.hru_t before searching all groups for *.hru_t, i.e. after their suffix has been removed.