COSTA
common set of tools for the assimilation of data

MEMO	CTA memo200605
Date	25-09-2006
Author(s)	Nils van Velzen
Subject	Using the modelbuilders in COSTA

Table of contents

1 Introduction

The COSTA environment makes a number of building blocks available for creating data assimilation and calibration systems. Combining and building new building blocks should be possible with a minimum of effort.

COSTA contains tools for rapidly creating a COSTA model component. These tools are called modelbuilders. The various modelbuilders are described in this document.

2 General description of a model

COSTA deals with assimilation methods for simulation models. Simulation models can compute the model state at different time instances.

$$\begin{aligned} \phi(t_0) &= \phi_0, \\ \phi(t_{i+1}) &= A[\phi(t_i), u(t_i), g] \end{aligned} \tag{1}$$

with

- ϕ_0 the initial model state,
- $\phi(t)$ the model state at time t,
- A the operator that computes one time-step of the numerical simulation model,

- $u(t)$ the time dependent forcings at time t,
- g the time independent model parameters

The model as stated in Equation 1 is a general form. This means that it is not mandatory that all arguments exist in the model. An extreme example is the model, as specified by Equation 2 that can be used in a calibration context where an optimal value for g is determined using observed data.

$$\phi = A[g] \tag{2}$$

3 SP Model builder

The SP modelbuilder (Single processor) can be used to create sequential (non-parallel) model components. The SP modelbuilder handles the storage and administration of the model-instance specific data. By using this modelbuilder it is possible to create a full working COSTA model component by only implementing a very small number of routines.

The routines that are supported in the current version of the SP Model builder are:

- cta_model_create
- cta_model_free (not yet supported)
- cta_model_compute
- cta_model_setstate
- cta_model_getstate
- cta_model_axpymodel
- cta_model_axpystate
- cta_model_setforc (not yet supported)
- cta_model_getforc (not yet supported)
- cta_model_axpyforc
- cta_model_setparam (not yet supported)
- cta_model_getparam (not yet supported)
- cta_model_axpyparam (not yet supported)
- cta_model_getnoisecount
- cta_model_getnoisecovar

- cta_model_getobsvalues
- cta_model_getobsselect
- cta_model_addnoise

Not all methods are supported in the current release of the modelbuilder. The modelbuilder will support in the near future however.

Using the modelbuilder the model programmer only needs to implement a small number of subroutines. The modelbuilder will use these subroutines for implementing all methods. The subroutines that must be provided by the model programmer and their interface are given in the following sections.

3.1 Create a new model instance

This routine creates and initialises a new model instance.

`USR_CREATE(hinput, state, sbound, sparam, nnoise,`
`time0, snamnoise, husrdata, ierr)`

`IN`	`hinput`	Model configuration CTA_Tree of CTA_String
`OUT`	`state`	Model state (initialized to initial value Note this statevector must be created
`OUT`	`sbound`	State-vector for the offset on the forcings. CTA_NULL if not used Note this statevector must be created
`OUT`	`nnoise`	The number of noise parameters in model state is 0 in case of a deterministic model
`OUT`	`time0`	Time instance of the initial state state The time object is already allocated
`OUT`	`snamnoise`	Name of the substate containing the noise parameters The string object is already allocated
`OUT`	`husrdata`	Handle that can be used for storing instance specific data
`OUT`	`ierr`	Return flag CTA_OK if successful

```
void usr_create(CTA_Handle *hinput,  CTA_State *state, CTA_State sbound,
                CTA_State *sparam, int *nnoise, CTA_Time time0,
                CTA_String *snamnoise, CTA_Handle *husrdata, int *ierr)

USR_CREATE(hinput, state, sbound, sparam, nnoise, time0,
           snamnoise, husrdata, ierr)
integer hinput, state, sbound, sparam, nnoise, time0
integer snamnoise, husrdata, ierr
```

3.2 Compute

This routine is computes several timesteps over a giving timespan.

```
USR_COMPUTE(timespan,state, saxpyforc, baddnoise, sparam, husrdata, ierr)
```

IN	timespan	Timespan to simulate
IN/OUT	state	State vector
IN	saxpyforc	Offset on models forcings
IN	baddnoise	flag (CTA_TRUE/CTA_FALSE) whether to add noise
IN	sparam	Model parameters
IN/OUT	husrdata	Instance specific data
OUT	ierr	Return flag CTA_OK if successful

```
void USR_COMPUTE(CTA_Time *timespan, CTA_State *state, CTA_State *saxpyforc,
                 int *baddnoise, CTA_State *sparam, CTA_HAndle *husrdata,
                 int *ierr)

USR_COMPUTE(timespan,state, saxpyforc, baddnoise, sparam, husrdata, ierr)
integer timespan,state, saxpyforc, baddnoise, sparam, husrdata, ierr
```

3.3 Covariance matrix of the noise parameters

This routine is responsible for returning the covariance matrix of the noise parameters.

```
USR_COVAR(colsvar,nnoise, husrdata, ierr)
```

OUT	colsvar(nnoise)	covariance of noise parameters array of noise Note the substates are already allocated
IN	nnoise	Number of noise parameters
IN/OUT	husrdata	Instance specific data
OUT	ierr	Return flag CTA_OK if successful

```
void usr_covar(CTA_State *colsvar, int *nnoise, CTA_Handle *husrdata, int *ierr)

USR_COVAR(colsvar, nnoise, husrdata, ierr)
integer nnoise, husrdata, ierr
integer colsvar(nnoise)
```

3.4 Model state to observations

This routine is responsible for the transformation of the state-vector to the observations.

```
USR_OBS(state, hdescr, vval, husrdata, ierr)
```

IN	state	state vector
IN	hdescr	Observation description of observations
OUT	vval	Model (state) values corresponding to observations in hdescr
IN/OUT	husrdata	Instance specific data
OUT	ierr	Return flag CTA_OK if successful

```
void usr_obs(CTA_State *state, CTA_ObsDescr *hdescr, CTA_Vector *vval,
             CTA_Handle *husrdata, int *ierr)

USR_OBS(state, hdescr, vval, husrdata, ierr)
integer state, hdescr, vval, husrdata, ierr
```

3.5 Observation selection

This routine is responsible for producing a selection criterion that will filter out all invalid observations. Invalid observations are observations for which the model cannot produce a corresponding value. For example observations that are outside the computational domain.

```
USR_OBSSEL(state, ttime, hdescr, sselect, husrdata, ierr)
```

IN	state	state vector
IN	ttime	timespan for selection
IN	hdescr	observation description of all available observations
OUT	sselect	The select criterion to filter out all invalid observations
IN/OUT	husrdata	Instance specific data
OUT	ierr	Return flag CTA_OK if successful

```
void usr_obssel(CTA_State *state, CTA_Time *ttime, CTA_ObsDescr *hdescr,
        CTA_String *sselect, CTA_Handle *husrdata, int* ierr)

USR_OBSSEL(state, ttime, hdescr, sselect, husrdata, ierr)
integer state, ttime, hdescr, sselect, husrdata, ierr
```

3.6 xml-configuration

The modelbuilder need to be configured in order to create a new model. This configuration specifies which functions are provided to implement the model.

The configuration has the following form (in xml)

```
<modelbuild_sp>
<functions>
   <!-- The functions that implement the model -->
   <create>my_create</create>
   <covariance>my_covar</covariance>
   <getobsvals>my_obs</getobsvals>
   <compute>my_compute</compute>
   <getobssel>my_getobssel</getobssel>
   <model>
   <!-- Everything overhere is passed through to the model (input argument hinput of
   </model>
</functions>
</modelbuild_sp>
```

This configuration file is read into a COSTA-tree and is used as input argument for each instance that is created.

The names of the functions eg. my_compute, correspond to the name specified when administrating the function in COSTA using the `cta_func_create`.

Future versions of the modelbuilder will support dynamic linking to the user functions. When this is supported it will be possible to directly link the routines from the dynamic link library.

3.7 Examples

The modelbuilder is used for the models lorenz96, lorenz, and oscill in the COSTA model-directory. These models are a source of information concerning the use of this modelbuilder.