# Introduction

This memo describes the concept and the configuration in Delft-FEWS to reduce the amount of model states in database. With default configurations, model states have an expiry time similar to the update module that generated them. However, if these update modules use warm state search periods prior to T0 and are scheduled once day, the majority of those states, the so called transient states, don’t need longer expiry days than 1 day.

For completeness reasons, this memo also describes general state handling in FEWS.

[1 Introduction 1](#_Toc33100858)

[2 General State Handling in Delft-FEWS 2](#_Toc33100859)

[2.1 State Selection 2](#_Toc33100860)

[2.2 State Saving 2](#_Toc33100861)

[3 Customized State Handling 3](#_Toc33100862)

[3.1 UpdateStates Workflows 3](#_Toc33100863)

[3.2 Forecast Workflows 4](#_Toc33100864)

[3.3 ESP Workflows 5](#_Toc33100865)

[4 Transient states 5](#_Toc33100866)

# General State Handling in Delft-FEWS

Delft-FEWS stores model states in the native model format. Cold states, default initial conditions, are stored in a zip file in the Config/ColdStateFiles directory. Warm States, initial conditions from a previous run, are stored in zip files in the Delft-FEWS database with a time stamp and module instance reference.

## State Selection

The State selection configuration defines which state is exported from FEWS to the model directory when running a FEWS workflow. As a result, the State selection determines the start of the simulation. In operational forecasting, usually a warm state is configured as the default setting in the search period. If no warm state is found, a cold state will be used at the beginning of warm state search period.

The default warm state selection period may be set in the Topology, in the General Adapter module, in the WorkflowDescriptors and in the TaskRunProperties in the Admin Interface. The order of precedence for these is generally the more granular wins: Topology beats ModuleConfigFile beats WorkflowDescriptors beats TaskRunProperties. In the case of nested workflows the top level workflow will win over the nested workflows. The configuration for the state selection looks as follows:

*<stateSelection>
            <warmState>
                <stateSearchPeriod start="-15" end="-10" unit="day"/>
            </warmState>*

*</stateSelection>*

In the Topology.xml this often configured as:

*<warmState unit="day" multiplier="10"/>*

The configured state selection period can be overruled in the GUI.

## State Saving

By convention, only the UpdateStates workflows save warm states in the FEWS database, at the end of the simulation run. This is configured in the importStateActivity of the General Adapter modules. The end of the simulation is determined by the end of the forecast length, which is typically set to T0 in UpdateStates workflows.

There are two options that will save states for other periods, although under the hood workflows are broken down into multiple simulation periods to achieve that.

* *loopTimestep*: When specified all activities are run in a loop to ensure on every cardinal time a state is produced between the exported state time and T0. This has two advantages. States are equally and frequently distributed over time so it is possible to start an UpdateStates workflow from every point. The other advantage is the restriction on memory consumption, one can run over months without going out of RAM.
* *writeIntermediateState*: When specified an extra state at end of the state search period is produced that can be used in the next state run.

# Customized State Handling

In several Delft-FEWS applications, notably the CHPS applications in the USA, there are separate workflows for the historical or UpdateState model execution, Forecast execution and Ensemble forecast execution..

## UpdateStates Workflows

Figure 1 represents the UpdateState workflow in CHPS. The run starts at the end of the warm state search period, see the blue solid square.

Figure 1, State handling in UpdateStates workflows in CHPS

UpdateState workflows:

* are scheduled at the server, once a day (midnight) with T0 at 12z
* have a warm state search period of T0 -15 days till T0-10 days and run till T0, as configured in the workflowDescriptor file.
* generate a state every day because of the loopTimeStep (12z) in the General Adapter
* are not defined in the Topology panel
* are linked to the Forecast run via the *simulatedHistoricalModuleInstanceId* option in the ModuleInstanceDescriptors

## Forecast Workflows

Figure 2 represents the Forecast workflow in CHPS. Also here the run starts at the end of the warm state search period, or just at T0-10days if this is configured as such in the Topology.xml.

Figure 2, State handling in Forecast workflows in CHPS

Forecast workflows:

* can run at any cardinal time step
* have a warm state search period of T0 -15days till T0-10 days, as configured in the workflowDescriptor file, and run till the forecast length
* do not generate states
* are defined in the Topology panel where each node synchronises with basin specific predefined plots and modifier display

Modifiers applied to the Forecast workflow (period before T0) in Topology panel are picked up by the UpdateStates workflow.

## ESP Workflows

Figure 2 represents the ESP workflow in CHPS.

Figure 3, State handling in ESP workflows in CHPS

Forecast ESP workflows:

* can run at any cardinal time step
* have a warm state search period of T0-10 till T0 and run till the forecast length. (So they usually start at T0)
* do not generate states
* are defined in the Topology panel where each node synchronises with basin specific predefined plots and modifier display
* modifiers from the deterministic runs can be excluded from the ESP run by configuration

# Transient states

When comparing Figure 1 and 2, it can be concluded that the UpdateStates workflow produces many states, but most of them are only important for the (ESP) Forecasts for a day. The next day, when the UpdateStates workflow generates new states, those previous states between the end of the warm state search period and T0 are not needed anymore. However they are still in the database and they will expire when the UpdateState workflow will expire. In other words, the size of the database is increased unnecessarily with many useless long living states. We call the states that are imported after the state search period the *transient* states.

Only the states in the warm state search period should be long living states. There are options in the state selection configuration to reduce the amount of space in the warm states tables of the database:

* **searchForTransientStates:** By default this option is true, but when setting to false, it prevents your transient states become part of your long living state branch. **transientStateExpiryTime:** In the state selection you can configure the transient state expiry time, for example 1 day, and the task run expiry time.  For the long living states the expiry time of the state run is used. Remember states cannot live longer than the update run that creates them.

So in the NWS UpdateStates workflow the searchForTransientStates in the state selection should be false so the new long living state is connected to another long living state. See Figure 4 for a schematic representation.

Figure 4, State handling when using transient states

Please note that if you compare Figure 4 with Figure 1, the UpdateState run will start one day earlier (T0-11days), because the state at T0 before execution is a transient state, produced by a previous UpdateState run.

An example of the UpdateStates workflow configuration in the workfowDescriptor file is as follows:

*<workflowDescriptor id="UpdateStates" forecast="true" visible="true" allowApprove="true"   autoApprove="true">
        <description>UpdateStates</description>
        <cardinalTimeStep id="12Z"/>
        <stateSelection>
            <warmState>
                <stateSearchPeriod start="-15" end="-10" unit="day"/>****<searchForTransientStates>false</searchForTransientStates>
                    <transientStateExpiryTime multiplier="1" unit="day"/>***  *</warmState>
        </stateSelection>
        <runExpiryTime multiplier="30" unit="day"/>
    </workflowDescriptor>*