This extra adds a new command which allows one to save some of the current state of a model into a file that provides a model definition which can be loaded and run.

This is the information that it records from the model:

- chunk-types that were defined by the model (any that are not defined by default)
- the current declarative memory chunks and their corresponding parameters
- the current productions and their parameters
- general parameters related to declarative and procedural modules
- chunks that are in buffers

That does not capture all of the current state because it does not include internal information from modules (like finsts or previously prepared motor features), events currently on the meta-process queue, nor any other commands or settings which were in the original model.

These are the general parameters which are written out if the current value differs from the default value:

DECLARATIVE module

- :MD
- :RT
- :LE
- :MS
- :MP
- :PAS
- :MAS
- :ANS
- :BLC
- :LF
- :BLL

CENTRAL-PARAMETERS module

- :ESC

- :ER
- :OL

UTILITY module

- :IU
- :UL
- :ALPHA
- :UT
- :NU
- :EGS

PRODUCTION-COMPILATION module

- :EPL
- :TT

PROCEDURAL module

- :DAT
- :PPM

It will also write out the current :seed parameter so that it could be used to reproduce the same results after the save point as the model which was saved. By default the :seed parameter will be included in a comment, but that can be changed when calling the command.

The appropriate declarative parameters will be recorded using calls to sdp. Which parameters to record are based on the current :mp, :bll, and :ol parameter settings. The possible parameters recorded are :creation-time, :reference-count, :reference-list, and :similarities. It does not record the current Sji values. The assumption is that they are dynamic based on the fans of the items. Thus, when the model is loaded it will restore those automatically and they will continue to change as new chunks are added to DM. If explicit values are set in the model then those would have to be added to the saved model file.

When writing out the creation-time and reference-list the times will be adjusted to a zero reference since the model time will be back to 0 when reloaded. If one wants to avoid the re-referencing of those times then there is a parameter that can be specified to prevent that.

For productions the :u and :at values will always be written out. Also, any non-nil :reward settings will be saved if utility learning is enabled. Those will be set using calls to spp. If production compilation is enabled then an additional production parameter, which is not available through spp, will be set to indicate which productions were learned by the model. That is done because the utility learning differs between productions which are defined explicitly in the model vs those learned through compilation when a newly composed production is a semantic match to an existing production. Without that setting all of the productions would be marked as explicitly defined when the model is loaded.

```
;;;
;;; save-chunks-and-productions
;;;
;;; (defun save-chunks-and-productions (file-name &optional (zero-ref t))
;;;
;;; This function takes one required parameter which is a string that names
;;; a file to create and write the saved model definition to.  If the optional
;;; parameter is specified as nil then the current reference times for the
;;; declarative parameters are written out exactly as they are.  If it is not
;;; provided or has any non-nil value then those times will be adjusted so that
;;; they correspond to a current time of 0.0s (the default time when the model
;;; is later loaded).
;;;
;;; The save-chunks-and-productions function is included for backward
;;; compatibility, but it is depricated and the new (as of v4.0a1) save-model-file
;;; function should be used instead.
;;;
;;;
;;;
;;; save-model-file
;;;
;;; (defun save-model-file (file-name &key (zero-ref t)
;;;                                        (comment-seed t)
;;;                                        skip-buffers
;;;                                        pre-chunk-type-hook
;;;                                        pre-chunk-hook
;;;                                        pre-production-hook
;;;                                        end-hook)
;;;
;;; The save-model-file writes out the model file in the same way as the
;;; save-chunks-and-productions function described above, but with some more
```

;;; options to adjust what gets written.
;;;
;;; The comment-seed parameter can be used to control whether the setting of the
;;; :seed parameter written to the file is commented out or not. The default
;;; is t (comment the setting).
;;;
;;; The skip-buffers parameter can be specified as a list of buffer names, and
;;; any buffer in that list will not have a line in the model to set its content
;;; if it had any at the time of saving. Note however that if there was a chunk
;;; in the buffer that chunk will still be created in the model since it could
;;; be referenced elsewhere.
;;;
;;; The four hook parameters can be used to write additional code into the model
;;; definition at the points indicated by the hook's name. The parameters can
;;; be specified as any valid command identifier and should output the code
;;; using the command-output command so that it is written into the file
;;; appropriately.

## save-model-file

### Syntax:

**save-model-file** *file-name* {**:zero-ref** *zero-ref*} {**:comment-seed** *comment-seed*}
{**:skip-buffers** *skip-buffers*}
{**:pre-chunk-type-hook** *chunk-type-hook*}
{**:pre-chunk-hook** *chunk-hook*}
{**:pre-production-hook** *production-hook*}
{**:end-hook** *end-hook*}
-> **nil**

### Remote command name:

**save-model-file** *file-name* { < **zero-ref** *zero-ref,* **comment-seed** *comment-seed,* **skip-buffers** *skip-buffers,*
**pre-chunk-type-hook** *chunk-type-hook,* **pre-chunk-hook** *chunk-hook,*
**pre-production-hook** *production-hook,* **end-hook** *end-hook* > }

#### Arguments and Values:

file-name ::= a string containing a path for a file to write the model to
zero-ref ::= a generalized boolean indicating whether times should be re-referenced to 0 (default true)
comment-seed ::= a generalized boolean indicating whether to comment the seed parameter setting
(default is true)
skip-buffers ::= (buffer-name*)
buffer-name ::= a symbol naming a buffer
chunk-type-hook ::= a command identifier specifying a command to write additional info to the model
chunk-hook ::= a command identifier specifying a command to write additional info to the model

production-hook ::= a command identifier specifying a command to write additional info to the model
end-hook ::= a command identifier specifying a command to write additional info to the model

**Description:**

This command will set the current model's :cmdt parameter to the file-name provided. Then it will write out the current model's information, as described above, using the command-output command. The hook parameters can be used to write additional information to the file. Once the saving is done the :cmdt parameter will be returned to the value that it had prior to calling this command. Because this command is using the :cmdt parameter and records the model's current state, for stability purposes, the model should not currently be running when this is called. If the zero-ref parameter is true then all of the recorded times for the chunk parameters will have the current time subtracted from them before writing them to the file. That way, if the model is loaded at time 0 then all of the activations for the chunks will be as they were when it was saved. If the comment-seed parameter is true then the model's current seed setting will be written in a comment, otherwise it will be set in the model definition. If skip-buffers is a list of buffer names then those named buffers will not have a command in the saved model to set their current chunk. It does not prevent the model from creating the chunk because that chunk could be referenced in other buffer chunks which are being restored. The hook parameter commands, if provided, will be called with no parameters at the indicated times in the writing of the model definition. They can use the command-trace command to output additional information into the file. They should always write syntactically correct model code because otherwise the file may fail when it is attempted to be loaded later.

The command always returns **nil** and currently has very little in the way of error checking which means invalid parameters are likely to cause errors instead of warnings.

Below is a skeleton of what the saved model file will look like, indicating where the hook commands are called. Items in < > indicate output that will be based on the indicated information and items in *{}* are describing how the provided parameters may adjust the content. A super-scripted * indicates that zero or more of the indicated item may occur.

```
;;; Saved version of model <model name> at run time <model time in sec> on <date and time>
(clear-all)
(define-model <model name>-saved
(sgp
```

```
<parameter and value>*

)

{if the comment-seed parameter is true there will be semicolons before this sgp line}

(sgp :seed (197174297069 0))


{output from the pre-chunk-type-hook command will appear here}

(chunk-type <user-defined-type>
)*

{output from the pre-chunk-hook command will appear here}

(define-chunks
<current declarative memory and buffer chunk definitions>*
)

(add-dm-chunks
   <chunk name of a chunk to add to DM from above>*
)

(sdp <chunk-name> <parameters and values>*
)*


{output from the pre-production-hook command will appear here}

(P <production definition>
)*


(spp <production-name> <parameters and values>*
)*


{a set-buffer-chunk for all buffers with chunks not on the skip-buffer lists except goal}

(set-buffer-chunk <buffer-name> <chunk-name>)*

{if the goal buffer has a chunk and isn't on the skip-buffers list set with goal-focus}

(goal-focus <chunk-name>)

{output from the end-hook command will appear here}
)
```