El formato `.dsa`

La especificación, la gramática y las guías

Diego Saravia
Facultad de Ciencias Exactas – Universidad Nacional de Salta
INENCO (UNSa – CONICET)

17 de noviembre de 2026

Resumen

El `.dsa` es el formato con que se describen los modelos de Simusol: el MODELO, los DATOS y los USOS de un circuito. Lo lee slex (`slex/dsa`, la biblioteca `libdsa`), con su gramática `dsa.slex`, y lo revisa `dsa-verifica`. Este documento reúne todo lo escrito sobre el formato. Cada parte era un documento en Markdown y está entera, con sus tablas, sus ejemplos y sus enlaces: los enlaces entre las partes son referencias internas, y los que llevan a otros archivos del árbol muestran su ruta.

Parte	Era
La gramática del .dsa de un pantallazo	nueva (17/11/2026): sale del lector y está probada con `dsa-verifica`
El lenguaje dsa: modelo general y modelos de elementos	`AcuerdosGenerales_dsa.md`, en la raíz de siru
Modelos de elementos en dsa	`ModelosDeElementosEn_dsa.md`, en la raíz de siru
Gramática .ds y .d	`GRAMATICA.md`, en `slex/d2datdinc/doc`
Ecuaciones con retardo en .dsa	`RETARDOS_DSA.md`, en `siru/doc`
DSA como flujo entre programas	`FLUJO_DSA.md`, en `siru/doc`
Quiero construir y utilizar un modelo	`COMO-ESCRIBIR-UN-MODELO.md`, en la raíz de siru

Índice

1. La gramática del .dsa de un pantallazo **1**
1.1. El archivo, entero 1
1.2. Las formas de renglón 2
1.3. Fuera de todo bloque 3
1.4. El modelo: MODELO y sus bloques 3
1.5. Los campos 4
1.6. Modelos de elementos y tablas 5
1.7. Los datos y los usos: DATOS, RESULTADOS, USOS 5
1.8. La familia SCEPTRE 6
1.9. Lo que el lector rechaza 7
1.10. Lo que el lector no controla 7

2. Lenguaje dsa: modelo general y modelos de elementos **8**
2.1. 1. Modelo, datos y usos . . . 8
2.1.1. Campos opcionales y descripción . . . 8
2.1.2. USOS: operaciones y análisis sobre el modelo . . . 9
2.1.3. Pedido POLOS_CEROS . . . 10
2.1.4. Secuencia M, DU y R por entrada y salida estándar . . . 11
2.2. 2. Contenido del modelo y origen de sus variables . . . 13
2.3. 3. Nombres, parámetros y derivadas . . . 14
2.3.1. Variable de integración y derivadas . . . 15
2.4. 4. Estados y dimensión del modelo . . . 15
2.4.1. Identificación explícita de estados del elemento . . . 17
2.5. 5. Números, unidades y resultados . . . 17
2.5.1. Unidades solicitadas en resultados . . . 18
2.6. 6. Métodos de integración . . . 19
2.7. 7. Modelos de elementos: relaciones y nombres locales . . . 19
2.7.1. Nombres internos y externos . . . 19
2.8. 8. Circuito, terminales y nombres de instancias . . . 20
2.8.1. Índices y nombres de elementos anidados . . . 20
2.8.2. Terminales y signos . . . 20
2.8.3. Valor indicado en la rama . . . 21
2.9. 9. Composición mediante ramas internas . . . 21
2.10. 10. Funciones y tablas locales . . . 22
2.10.1. Funciones dentro de un modelo: ejemplo completo . . . 22
2.10.2. TABLE y opciones de Tabular . . . 23
2.10.3. Tablas dentro de un modelo: ejemplo completo . . . 23
2.10.4. Función que usa una tabla local . . . 24
2.11. 11. Catálogo de ejemplos de elementos . . . 24
2.11.1. Elementos de cuatro terminales . . . 25
2.12. 12. Lazos, cortes y responsabilidad del generador . . . 26
2.12.1. Condensador: ecuaciones propias del elemento . . . 26
2.12.2. Ejemplo de condensadores en un corte (cutset) . . . 27
2.12.3. Ejemplo de lazo de condensadores y ecuación derivada . . . 27
2.12.4. Qué pertenece al elemento y qué pertenece al esquema general . . . 28
2.12.5. Marcas LOOP y CUTSET . . . 28
2.12.6. Condiciones matemáticas: lógica interna del programa . . . 29
2.13. 13. Llaves y cambios de configuración . . . 30
2.13.1. Opción implementada: llaves resistivas (1/11/2026) . . . 30
2.14. 14. Elementos con memoria: convolución . . . 31
2.15. 15. Ejemplo completo: un modelo, datos y un cálculo puntual . . . 31
2.16. 16. Comparación con las bibliotecas actuales . . . 32
2.16.1. Resistencia: definición actual y propuesta . . . 33
2.16.2. Condensador: ecuaciones y estado . . . 33
2.16.3. Bobina y fuente: traducción de relaciones y signos . . . 34
2.16.4. Modelos con ramas internas: qué se conserva . . . 34
2.16.5. Lazos y cortes: dónde están hoy las ecuaciones adicionales . . . 35
2.17. 17. Cobertura del corpus y validación . . . 35
2.18. 18. Evaluación frente al esquema actual . . . 36
2.19. Formatos de entrada: dsa, y nada más . . . 37
2.20. 19. Qué queda por concretar e implementar . . . 38
2.21. 20. Registro de decisiones reemplazadas . . . 38
2.22. 21. Nota sobre organización de archivos . . . 39

2.23. 22. Referencias locales . . . 39
2.24. Selector de integración unificado . . . 39
2.25. Selector del generador de ecuaciones . . . 40
2.26. PARAPLOT en datos y usos . . . 40
2.27. Cadena completa del proyecto . . . 41
2.28. Motor y salida de Maxima (2026-11-06) . . . 42
2.28.1. Informe HTML de PARAPLOT . . . 42
2.29. 23. La lista completa de bloques y campos que el lector acepta . . . 42
2.29.1. 23.1 La familia SCEPTRE . . . 42
2.29.2. 23.2 La dinámica de siru . . . 43
2.29.3. 23.3 Las declaraciones de preparación y los metadatos . . . 44
2.29.4. 23.4 Reservados . . . 45
2.30. 24. El verificador . . . 45

3. Modelos de elementos en dsa **45**

4. Gramática .ds y .d **45**
4.1. 1. Un lenguaje compartido . . . 46
4.2. 2. Elegir el problema y ejecutarlo . . . 47
4.3. 3. `.d` o `.ds`: cuál usar . . . 47
4.3.1. 3.1 Estructura y reglas de escritura . . . 49
4.3.2. 3.2 Secciones del archivo . . . 51
4.3.3. 3.3 Gramática de ramas y modelos . . . 52
4.3.4. 3.4 Parámetros, incógnitas, constantes y ecuaciones . . . 53
4.3.5. 3.5 Funciones y tablas . . . 54
4.3.6. 3.6 Controles, condiciones y salidas . . . 55
4.3.7. 3.7 DATA y DATABLOB . . . 56
4.3.8. 3.8 FORMAS: el dibujo dentro del .ds . . . 57
4.3.9. 3.8.1 DEFAULTDOMAIN: dominio de las ramas sin forma . . . 58
4.3.10. 3.9 Unidades y resultados comunes; PARAPLOT para el dibujo . . . 61
4.3.11. 3.10 Cómo comprobar la lectura . . . 63
4.4. 4. Un archivo mínimo . . . 64
4.5. 5. Los elementos . . . 65
4.5.1. 5.1 La biblioteca (`slex/d2datdinc/biblioteca/libraryori.siru`) . . . 65
4.5.2. 5.2 Lo que un elemento puede nombrar en su valor . . . 67
4.5.3. 5.3 Tipos propios . . . 68
4.5.4. 5.4 Modelos de SCEPTRE (sólo `.d`) . . . 69
4.5.5. 5.5 Convolución: una caja descripta por su respuesta (`K`) . . . 70
4.6. 6. Llaves . . . 71
4.6.1. `anterior(X)`: la histéresis . . . 71
4.6.2. 6.1 Llaves o `if(...)`: la diferencia . . . 71
4.7. 7. Lazos de capacitores y cortes de bobinas . . . 72
4.8. 8. Acoplamientos magnéticos . . . 73
4.9. 9. Funciones y tablas . . . 73
4.10. 10. Ecuaciones propias, estados propios, parámetros . . . 76
4.11. 11. La corrida . . . 77
4.11.1. Composición de subflujos (SPLIT) . . . 78
4.11.2. GLM fijo o adaptativo . . . 79
4.11.3. Paso por error de truncamiento (el de SPICE) . . . 80
4.11.4. Desde dónde se escribe la tabla (`SALIDA_DESDE`) . . . 80
4.11.5. Partición explícita e implícita (PGLM / IMEX) . . . 81
4.11.6. Rosenbrock (extension ROS) . . . 81

4.11.7. Controles generales y Runge-Kutta . . . 82
4.12. 11 bis. El punto de operación (RUN INITIAL CONDITIONS) . . . 84
4.12.1. Estados que quedan como los da el archivo (IC_DADAS) . . . 84
4.13. 11 ter. Sensibilidad (RUN SENSITIVITY) . . . 84
4.14. 11 quater. Monte Carlo (RUN MONTE CARLO) . . . 85
4.15. 11 quinquies. Peor caso (RUN WORST CASE) . . . 85
4.16. 11 sexies. Reruns (RERUN DESCRIPTION) . . . 86
4.17. 11 septies. Seguir una corrida (CONTINUE, --continuar) . . . 86
4.18. 11 octies. El análisis en frecuencia (RUN AC) . . . 87
4.19. 11 nonies. Medidas sobre la corrida (MEDIDAS) . . . 87
4.19.1. Verificaciones (VERIFICACIONES) . . . 88
4.20. 11 decies. El espectro de una curva (ESPECTROS) . . . 89
4.21. 11 undecies. Curvas hechas de otras (CURVAS) . . . 89
4.22. 11 duodecies. El guion de vectores (GUION) . . . 89
4.23. 12. Un punto, sin integración: INTEGRA=[-1] . . . 90
4.24. 13. Los mensajes, y qué hacer . . . 92
4.25. 14. Lo que no se puede . . . 94
4.26. 15. PARAPLOT completo: fuentes, series, paneles y unidades . . . 95
4.26.1. 15.0 Grupos PARAPLOT y resultado GNUPLOT . . . 95
4.26.2. 15.1 Qué se dibuja y qué archivo contiene cada cosa . . . 95
4.26.3. 15.2 Todas las órdenes del motor histórico (v1) . . . 96
4.26.4. 15.3 Órdenes estructuradas (v2) . . . 98
4.26.5. 15.4 RESULTADOS, UNIDADES y conversiones . . . 99
4.26.6. 15.5 Un punto, varios estados y ausencia de tiempo . . . 100
4.27. 16. Resumen formal, alcance y archivos auxiliares . . . 100
4.28. 17. Ejemplos completos y comprobaciones . . . 102
4.28.1. Fuentes y mantenimiento de la referencia . . . 103
4.29. Motor y salida de Maxima (2026-11-06) . . . 103
4.29.1. Informe HTML de PARAPLOT . . . 104
4.29.2. Selección explícita y predeterminada de integración . . . 104

5. Ecuaciones con retardo en .dsa **104**
5.1. Alcance inicial . . . 104
5.2. Sintaxis . . . 104
5.3. Ejemplo 1: un retardo . . . 105
5.4. Ejemplo 2: varios retardos y estados . . . 106
5.5. Separación entre M y DU . . . 106
5.6. Selector y controles implementados . . . 107
5.7. Condiciones de parada — 07/11/2026 . . . 107

6. DSA como flujo entre programas **108**
6.1. Contrato acordado . . . 108
6.2. Estado de la implementación al 11 de noviembre de 2026 . . . 109

7. De SPICE al .dsa: lo que traduce cir2dsa **110**
7.1. El circuito . . . 110
7.2. Los análisis . . . 111
7.3. El guion (.control) . . . 112
7.4. El azar . . . 113
7.5. Lo que todavía corta . . . 114

8. Quiero construir y utilizar un modelo: qué escribir y cómo entregarlo 114
8.1. 1. Las tres cosas que tengo que proporcionar . . . 114
8.2. 2. Primero escribo el modelo del sistema . . . 115
8.2.1. Dónde escribo cada cosa en .ds . . . 117
8.3. 3. Después doy los datos del caso . . . 118
8.4. 4. Por último pido una operación . . . 118
8.4.1. 4.1 Si voy a simular: uso el método de integración del modelo . . . 119
8.4.2. 4.2 Qué escribo hoy para elegir un método . . . 120
8.4.3. 4.3 Si quiero definir un método propio . . . 121
8.5. 5. Cómo entrego las tres partes a cada programa . . . 122
8.5.1. 5.1 Calcula: modelo más archivo de casos . . . 122
8.5.2. 5.2 Siru: ejemplo completo con modelo, datos y operación identificados . . 123
8.5.3. 5.3 SCEPTRE: las tres partes reunidas en un .d . . . 123
8.5.4. 5.4 Puyuspa: dónde cargar el archivo . . . 124
8.5.5. 5.5 SPICE/ngspice: modelo, datos y análisis en una netlist . . . 125
8.6. 6. Antes de ejecutar, compruebo que entregué todo . . . 125
8.7. 7. Nuevo lenguaje: archivos .dsa . . . 126
8.8. Referencias para escribir y ejecutar . . . 127

1. La gramática del .dsa de un pantallazo

Todo el formato en pocas tablas: qué bloques hay, dónde va cada uno y qué renglones acepta. Es lo que el lector (`libdsa`, `dsa.c`) y su léxico (`dsa.slex`) aceptan de verdad: cada forma y cada ejemplo de esta parte pasó por `dsa-verifica` el 17 de noviembre de 2026. El significado de cada cosa está en las partes que siguen; acá sólo está la forma.

1.1. El archivo, entero

Un modelo son dos archivos: `X.M.dsa`, el MODELO, y `X.DU.dsa`, los DATOS y los USOS. Cada renglón es una sola cosa: un encabezado de bloque, un campo, una rama, una asignación... Todo bloque se abre con su nombre solo en el renglón y se cierra con `FIN`.

```
# dsa                          X.M.dsa: el modelo
MOTOR = siru                   (opcional, solo fuera de todo bloque)
DESCRIPCION: "texto"           (opcional)
MODELO_ELEMENTO Siru_X ... FIN (opcional: modelos de elementos propios)
TABLA "nombre" ... FIN         (opcional: tablas de datos)
MODELO
  VARIABLE_INTEGRACION: T      campos: declaraciones del modelo
  ESTADOS: VC1
  PARAMETROS: R, C
  CIRCUITO ... FIN             las ramas
  LLAVES ... FIN               las llaves
  ECUACIONES ... FIN           las ecuaciones
  FUNCIONES ... FIN            y los demas bloques del modelo
FIN
```

```
RESULTADOS                     X.DU.dsa: los datos y los usos
  T: VC1, IR1                  que se quiere ver
FIN
INTEGRA = ["112"]              el metodo (o NETLIST = ["mtk"])
DATOS                          un caso: los valores
  R = 1000
  TIEMPOFINAL = 5
```

```
FIN
USOS                          que hacer con el modelo y esos datos
  CONTROLES ... FIN
  CONDICIONES ... FIN
  SENSITIVITY ... FIN         y los demas analisis
FIN
PARAPLOT G1 ... FIN           (opcional: los graficos)
```

Varios `DATOS` seguidos de un `USOS` son varios casos con el mismo uso; un `RESULTADOS` vale para los casos que siguen.

1.2. Las formas de renglón

Son los tokens de `dsa.slex`: un renglón entero es de una de estas formas, y cada bloque dice cuáles acepta.

Forma	Cómo se escribe	Ejemplo
bloque	`NOMBRE` solo en el renglón, y `FIN` para cerrarlo	`CIRCUITO ... FIN`
campo	`NOMBRE: valor`	`ESTADOS: VC1, X`
rama	`NOMBRE: Modelo, nodo, nodo = valor`	`R1: Resistencia, 1, 2 = 1000`
llave	`SB-NOMBRE, nodo-nodo = condición`	`SB-S1, 1-2 = T>1`
asignación	`NOMBRE = valor`	`R = 1000`
función	`NOMBRE(argumentos) = expresión`	`DOBLE(x) = 2*x`
residuo	`expresión = 0`	`DX + K*X = 0`
ecuación con variables	`expresión V: variable, variable`	`IR1*R V: IR1, R`
conjunto	`(dependientes / independientes)`	`(VC1 / R, C)`
cadena	una cadena JSON	`"X PLOT DIMENSION = 0"`
objeto	un objeto JSON	`{"eq": "IR1 - IC1"}`
texto	cualquier otro renglón; lo interpreta el bloque	`T<TIEMPOFINAL`
comentario	`#` al principio; se ignora	`# una nota`

1.3. Fuera de todo bloque

Renglón	Qué es	Ejemplo
`MOTOR = nombre`	El motor del modelo. Sólo fuera de todo bloque; si se repite, con el mismo valor.	`MOTOR = siru`
`DESCRIPCION: cadena`	La descripción: una cadena JSON (con `\n` para varios renglones), `null` o vacía.	`DESCRIPCION: "Un RC"`
`INTEGRA = [..]`	El método de integración. Con `NETLIST`, las únicas asignaciones que van fuera de un bloque.	`INTEGRA = ["112"]`
`NETLIST = [..]`	Quién genera las ecuaciones de la netlist.	`NETLIST = ["mtk"]`

Renglón	Qué es	Ejemplo
`MODELO_ELEMENTO Nombre`	Bloque: el modelo de un elemento (abajo).	
`TABLA "nombre"`, `SIMUL "nombre"`	Bloque: una tabla de datos (abajo).	
`PARAPLOT nombre`	Bloque: un gráfico. Sólo fuera de todo bloque; su contenido se copia tal cual hasta `FIN`.	`PARAPLOT G1`
`MODELO`, `DATOS`, `RESULTADOS`, `USOS`	Los cuatro bloques principales.	

1.4. El modelo: MODELO y sus bloques

Directamente dentro de `MODELO` van campos, la marca `LINEALIZADO` y bloques; una asignación o un texto sueltos se rechazan.

Bloque	Renglones que acepta	Ejemplo
`CIRCUITO`	Ramas. El modelo es uno de los de biblioteca (abajo) o `Siru_TIPO`.	`C1: Condensador, 2, GND = C`
`LLAVES`	Llaves.	`SB-S1, 1-2 = T>1`
`ECUACIONES`	Residuos (con o sin `= 0`), ecuaciones con variables, asignaciones, y funciones igualadas a 0.	`IR1 - IC1 = 0`
`FUNCIONES`	Funciones.	`DOBLE(x) = 2*x`
`PARAMETROS_FIJOS`	Asignaciones: constantes.	`PI = 3.14159`
`SEMILLAS`	Asignaciones: valores iniciales de variables, para ayudar a converger.	`IR1 = 0.001`
`CONDICIONES_INICIALES`	Asignaciones `NOMBRE = valor`, o `NOMBRE = {"value": ..}`.	`VC1 = 2`
`DERIVADAS`	`D` + un estado ya declarado = `nombre`: la derivada tiene nombre propio.	`DVC1 = IC1`
`VALORES_RAMA`	`RAMA = [[valor, [variables]], ..]`: corrige el valor de una rama.	`R1 = [["(T<1)+(T>3)", ["T"]]]`
`ACOPLAMIENTOS`	`NOMBRE = {"bobinas": [..], "valor": .., "orden": n}`; `bobinas` (dos o más) y `valor` son obligatorios.	`M12 = {"bobinas": ["L1", "L2"], "valor": 0.5}`
`EXTERNAS`	`NOMBRE = {"argumentos": n, "fuente": "archivo"}`: una función externa.	`FFFF = {"argumentos": 3, "fuente": "f.f90"}`
`RETARDOS`	`ALIAS = ESTADO, retardo`. Exige la `HISTORIA` de cada estado.	`VRET = VC1, 1`
`HISTORIA`	`ESTADO = expresión`: la historia antes del tiempo inicial.	`VC1 = 0`

Bloque	Renglones que acepta	Ejemplo
DIFUSIONES	DIFUSION(estado, ruido) = expresión; el estado, declarado, y el par no se repite.	DIFUSION(VC1, W1) = 0.1
MECANICA	Campos COORDENADAS: y MOMENTOS: (igual cantidad, todos estados) y las funciones MASA_INVERSA(i,j), FUERZA(momento), RESTRICCION(k), GRADIENTE(k,coordenada). No va con CIRCUITO.	GRADIENTE(1,Q1) = 1
ECUACIONES_SIRU	Un objeto por renglón, sólo con las claves eq, vec, vars, indep, lineal, nucleo, res, tipo.	{"eq": "IR1 - IC1", "tipo": "N"}
METODOS_INTEGRACION	NOMBRE = JSON: datos de los métodos.	rk = {"orden": 4}
LLAVES_RESISTIVAS	NOMBRE = [RON, ROFF]; GENERAL vale para todas.	S1 = [0.001, 1e9] ([RON, ROFF], en ohmios)

Los modelos de biblioteca de una rama son Resistencia, ResistenciaRC, Condensador, CondensadorIncremental, Bobina, FuenteTensionSceptre, FuenteTensionSceptreEC, FuenteCorriente, FuenteCorrienteJC, ConvolucionImpedancia y ConvolucionAdmitancia; cualquier otro tiene que ser un Siru_TIPO declarado con MODELO_ELEMENTO.

1.5. Los campos

Campo	Valor, y dónde va	Ejemplo
VARIABLE_INTEGRACION:	Un nombre: la abscisa (por omisión, T).	VARIABLE_INTEGRACION: T
ESTADOS:	Lista de nombres.	ESTADOS: VC1, X
ESTADOS_IMPLICITOS:	Lista: estados cuya derivada ya es un parámetro. Antes que ESTADOS:.	ESTADOS_IMPLICITOS: X
PARAMETROS:	Lista: los datos que el modelo pide.	PARAMETROS: R, C
OBSERVABLES:	Lista: los nombres observables.	OBSERVABLES: VC1, IR1
VARIABLES:	Lista: las variables de análisis.	VARIABLES: VC1
RUTINA_SCEPTRE:	Un número de 1 a 4: TRAP, RUK, XPO, IMPLICIT.	RUTINA_SCEPTRE: 3
COORDENADAS:, MOMENTOS:	Listas. Sólo dentro de MECANICA.	COORDENADAS: Q1
FORMATO:	El formato de las filas. Sólo dentro de TABLA.	FORMATO: pares
SIRU:	Un objeto JSON. Sólo dentro de MODELO_ELEMENTO.	SIRU: {"tipo": "RX"}
LINEALIZADO	Marca sin valor. Sólo directamente dentro de MODELO.	LINEALIZADO

1.6. Modelos de elementos y tablas

Bloque	Renglones que acepta	Ejemplo
`MODELO_ELEMENTO` `Siru_X`	El campo `SIRU:` con sus metadatos (la clave `tipo` es obligatoria; además `valores`, `clase`, `if`, `ic`, `op`, `inv`) y las ecuaciones del modelo. `ESTADOS:` adentro es sólo declarativo.	`V - x1 * I = 0`
`MODELO_ELEMENTO` de un modelo de biblioteca	Se saltea entero hasta `FIN`: el modelo ya está en la biblioteca.	`MODELO_ELEMENTO` `Resistencia`
`TABLA "nombre"`	El campo `FORMATO:` y las filas, cada una una cadena JSON.	`"0 0"`
`SIMUL "nombre"`	Como `TABLA`, con el formato `simul`.	`"0 1 2"`

1.7. Los datos y los usos: DATOS, RESULTADOS, USOS

Bloque	Renglones que acepta	Ejemplo
`DATOS`	Asignaciones: el valor de cada parámetro y de cada estado, y `T` y `TIEMPOFINAL`.	`TIEMPOFINAL = 5`
`RESULTADOS`	Lista de salidas separadas por comas, cada una con su unidad opcional; `:` después de la primera la marca como abscisa.	`T: VC1, IR1`
`USOS`	Sólo bloques: los de abajo.	
`CONTROLES`	Asignaciones `CLAVE = [valor]`. `GLM`, `ROS`, `PGLM` y `SPLIT` eligen el método por su número.	`SIRU_PASO = [0.05]`
`CONDICIONES`	Condiciones de parada: texto, residuos, o funciones igualadas a 0.	`T<TIEMPOFINAL`
`SENSITIVITY`, `MONTE_CARLO`, `WORST_CASE`	Conjuntos, uno por renglón.	`(VC1 / R, C)`
`OPTIMIZATION`	Conjuntos, o renglones de SCEPTRE como cadenas.	`(VC1 / R)`
`POLOS_CEROS`	Conjuntos `(salida / fuente)`.	`(VC1 / E1)`
`MEDIDAS`	`NOMBRE = JSON`: una medida sobre la corrida (máximo, promedio, cruce, demora…); siru la calcula al terminar.	`vmax = {"tipo":` `"maximo", "de": "U2"}`
`VERIFICACIONES`	`NOMBRE = JSON`: una condición que la corrida tiene que cumplir, con las medidas y las columnas de la tabla; siru dice si se cumple.	`carga = {"condicion":` `"abs(vmax - 5) < 0.01"}`

Bloque	Renglones que acepta	Ejemplo
ESPECTROS	NOMBRE = JSON: la transformada de Fourier de una curva de la corrida; siru la deja en NOMBRE.fft.	e1 = {"de": "U1", "paso": 2e-6}
RUIDO	salida, entrada y fuentes, cada uno = JSON: el análisis de ruido; siru lo deja en NOMBRE.ruido.	entrada = "v1"
GUION	pNNNN = "renglón": el guion de vectores, un programa que se corre sobre la tabla de resultados (álgebra de las curvas: elementos, bucles, transformadas).	p0001 = "let h = sum(T > 26)"
CURVAS	NOMBRE = JSON: una curva hecha de otra después de la corrida (promedio móvil, derivada, integral), a NOMBRE.curvas.	f = {"tipo": "derivada", "de": "U1"}
COTAS	NOMBRE = JSON: la cota de un parámetro o elemento.	R = {"min": 900, "max": 1100}
AC	NOMBRE = [real, imaginaria]: el valor complejo de una fuente.	E1 = [1, 0]

1.8. La familia SCEPTRE

Conservan lo que un deck .d dice y el dato de siru no modela, para que la vuelta .d → .dsa → .d sea completa.

Bloque	Renglones que acepta	Ejemplo
CONTROLES_SCEPTRE	Cadenas: cada renglón de RUN CONTROLS, tal cual.	"X PLOT DIMENSION = 0"
ELEMENTOS_SCEPTRE	Cadenas: renglones de ELEMENTS que sólo SCEPTRE entiende.	"T1, B-C-E = MODEL 2N706A"
SALIDAS_AC	SALIDA = {objeto}: la marca AC y la presentación de cada salida.	VC1 = {"ac": "D", "rotulo": "U2"}
SALIDAS_TARJETAS	Un objeto por renglón: cada tarjeta de OUTPUTS, tal cual.	{"vars": [["VC1", "X"], ["IR1"]]}
RERUNS	Un objeto por renglón: una corrida repetida, con sus cambios.	{"n": 2, "cambios": [..]}

1.9. Lo que el lector rechaza

Un bloque o un renglón que el lector no sabe leer corta nombrándolo, con su renglón: nada se omite en silencio.

Qué	Por qué
OPERACION, FORMULAS, OPCIONES_SPICE, COMANDOS_SPICE	Están en el léxico como nombres de bloque, pero el lector todavía no los lee: reservados.

Qué	Por qué
`INICIO_SALIDA:`, `INICIALIZACION:`	Están en el léxico como campos y la especificación los nombra (sección 23.3), pero el lector los rechaza.
Una asignación o un texto directamente en `MODELO` o en `USOS`	Esos bloques sólo llevan campos y otros bloques.
Una asignación fuera de todo bloque	Salvo `INTEGRA` y `NETLIST`.
Un modelo de rama que no es de biblioteca ni `Siru_TIPO`	El lector no sabe qué elemento es.
`MOTOR` o `PARAPLOT` dentro de un bloque; `LINEALIZADO` fuera de `MODELO`	Tienen un solo lugar.
Un valor que tenía que ser JSON y no lo es	En `COTAS`, `AC`, `ACOPLAMIENTOS`, `EXTERNAS` y los demás bloques de JSON.
`RETARDOS` sin la `HISTORIA` de cada estado	Sin historia no hay con qué empezar.

1.10. Lo que el lector no controla

El lector principal sólo mira el bloque más interno, así que no comprueba que cada bloque esté dentro del que le corresponde: un `CIRCUITO` fuera de `MODELO` se lee igual. Tampoco avisa si falta el `FIN` del último bloque. La pasada que separa los casos del `X.DU.dsa` sí es estricta: en ese archivo sólo admite `DATOS`, `RESULTADOS`, `USOS`, `INTEGRA` y `NETLIST` fuera de un bloque, y dentro de `USOS` sólo los bloques de la tabla de arriba. Qué claves de `CONTROLES` valen lo decide después quien usa el dato, no este lector.

Para revisar un archivo: `dsa-verifica X.M.dsa` (sección 24 de la especificación).

2. Lenguaje dsa: modelo general y modelos de elementos

Especificación de los acuerdos de diseño. Reúne el modelo general y los modelos de elementos en un solo documento. Distingue reglas acordadas, ejemplos ilustrativos y aspectos que requieren implementación o validación. Los modelos actuales se citan únicamente en la comparación; su sintaxis no se confunde con la propuesta dsa.

2.1. 1. Modelo, datos y usos

2.1.1. Campos opcionales y descripción

Todos los campos y secciones son opcionales para la lectura. Se pueden leer archivos parciales —por ejemplo, un DU antes de disponer del modelo— e incluso un archivo vacío. La ausencia de información no es un error de lectura: al preparar el cálculo, la simulación o una traducción que necesite esa información, se comprueba que el conjunto reunido sea suficiente y coherente. Esto no elimina la comprobación de sintaxis de lo que sí se escribió, como los cierres FIN o la estructura de los bloques.

`DESCRIPCION` es un campo opcional de texto descriptivo; no introduce variables ni ecuaciones. Se escribe, por ejemplo, `DESCRIPCION: "Circuito de prueba"`. Se admite su ausencia, `DESCRIPCION:` sin contenido y `DESCRIPCION: ""`. El lector también interpreta el `DESCRIPCION: null` de conversiones anteriores como vacío. El escritor omite el campo cuando no hay descripción. En la separación M/DU, la descripción del sistema queda en M.

RESULTADOS queda fuera de MODELO. Selecciona las variables que se desea informar en cada uso, con unidades de salida opcionales; puede incluir parámetros conocidos y variables calculadas. Cambiar esa selección no cambia el modelo, no crea variables ni agrega ecuaciones. Pedir que se informe una variable tampoco la convierte en dato: imponer su valor corresponde a DATOS. El mismo modelo puede utilizarse con diferentes datos, operaciones y selecciones de resultados.

RESULTADOS es un bloque separado de MODELO. El traductor lo escribe en el archivo DU y conserva la lectura de la ubicación antigua dentro de MODELO.

La abscisa es la primera variable, por su posición (19/11/2026). No hace falta marcarla: en `RESULTADOS T, VCA, IR5` la abscisa es `T` porque está primera, y se puede poner otra variable en ese lugar si se quiere graficar contra otra cosa. Los dos puntos que escribían las conversiones anteriores –`T: VCA,IR5`– están de más y el escritor ya no los pone; el lector los sigue aceptando, porque los archivos convertidos los traen. La abscisa del gráfico no es la variable de integración: coinciden casi siempre, pero son cosas distintas y se declaran en lugares distintos (`VARIABLE_INTEGRACION` en MODELO, la abscisa acá).

La variable de integración y sus valores son ortogonales (19/11/2026). `VARIABLE_INTEGRACION` dice *cuál* es la variable con la que se integra; su valor inicial y su valor final son datos, y van donde van los datos. Confundirlos lleva a hablar de "tiempo inicial" y "tiempo final" cuando la variable de integración no tiene por qué ser el tiempo.

Parte	Significado
Modelo del sistema	Describe variables, ecuaciones, parámetros fijos si los hay, semillas, funciones y modelos de elementos. Incluye las relaciones del método de integración cuando se incorpora uno.
Datos	Valores suministrados para cada uso del modelo: determinan un punto o las condiciones iniciales de una evolución.
USOS	Operaciones que se pide hacer con el modelo y esos datos: resolver un punto, buscar un equilibrio, calcular una evolución u otro análisis definido.

Invariante obligatorio de M/DU: DU aporta datos, pedidos y resultados, pero nunca modifica M ni vuelve a definir el `.dat`, los `.din` o las etapas de preparación del modelo en SCEPTRE. Esos productos se derivan de M; cambiar de DU no cambia su definición. Si se quiere otro integrador, se selecciona al preparar M —mediante su `INTEGRA` predeterminado o una sustitución explícita en esa etapa—, antes de aplicar DU. La implementación debe comprobar esta separación y rechazar un DU que intente cambiar el modelo o el integrador; no debe corregirlo silenciosamente.

`TIEMPOFINAL` pertenece a DU: su valor se escribe en `DATOS` y la condición de parada que lo utiliza, en `USOS / CONDICIONES`. No se declara en `PARAMETROS` de M. Cambiar el tiempo final no cambia el modelo ni exige volver a prepararlo. El tiempo final es irrelevante para armar M: su ausencia tampoco impide prepararlo. Los límites y sus valores por defecto se interpretan únicamente al ejecutar el uso. El traductor aplica esta separación a las entradas históricas y SPICE y al reescribir declaraciones antiguas.

El traductor separa `X.d` o `X.ds` en `X.M.dsa` (modelo) y `X.DU.dsa` (datos, usos y resultados). Ambos se pueden incorporar por separado o reunir mediante `cat X.M.dsa X.DU.dsa > X.dsa`; no hay una referencia obligatoria desde el modelo a un DU concreto.

Un mismo modelo puede utilizarse con distintos datos. Varios bloques de DATOS en el mismo archivo señalan varios casos: según el programa, se ejecutan como corridas separadas o como una ejecución múltiple. No es necesario organizar esa repetición desde fuera del archivo. Cada bloque aporta los datos de su caso; la operación solicitada determina qué se calcula con ellos. Una trayectoria tiene puntos relacionados por la evolución; varios bloques de datos no implican por sí solos una trayectoria ni que un caso continúe el anterior.

Las semillas forman parte del modelo como información numérica para facilitar el cálculo. Proporcionan aproximaciones iniciales para las incógnitas; el solver puede modificarlas al buscar la solución. Se distinguen de los DATOS de cada caso, que fijan valores o condiciones iniciales. Una semilla no impone una condición física ni agrega una ecuación. En una evolución, un dato inicial fija el estado al comienzo, no durante toda la trayectoria.

Generador: parte del programa que arma las ecuaciones a partir del modelo escrito. Expande las ramas mediante sus modelos de elementos e incorpora las relaciones de conexión, las del método de integración y las correspondientes a lazos y cortes. No es una sección que el autor deba escribir en .dsa.

2.1.2. USOS: operaciones y análisis sobre el modelo

`USOS` reúne lo que se solicita hacer con el modelo y los datos. Incluye los análisis de SCEPTRE, SPICE, Siru, Calcula u otros motores; no se limita al cálculo de un punto o de una evolución temporal. Un mismo modelo puede tener varios usos.

Uso	Qué solicita y qué opciones le corresponden
Punto de operación	Resolver un estado compatible con los datos y las condiciones indicadas.
Equilibrio estacionario	Buscar un estado estacionario, imponiendo las condiciones correspondientes sobre las derivadas.
Evolución temporal / transitorio	Calcular una trayectoria; indicar intervalo temporal y controles de ejecución numérica.
DC / barrido	Resolver puntos variando determinados datos o parámetros; indicar variables, rangos y pasos del barrido.
AC	Calcular la respuesta a pequeñas perturbaciones alrededor de un punto de operación; indicar excitaciones del análisis y frecuencias.
POLOS_CEROS	Calcular polos y ceros de la transferencia linealizada entre una entrada y una salida elegidas, alrededor de un punto estacionario.
SENSITIVITY	Calcular la sensibilidad de resultados respecto de datos o parámetros seleccionados; indicar dependientes, independientes y método.
OPTIMIZATION	Buscar valores que optimicen un objetivo; indicar variables de búsqueda, objetivo, cotas, restricciones y método.
WORST CASE	Buscar resultados extremos según las cotas y el método especificado; conservar qué significa peor caso en el motor de origen.

Uso	Qué solicita y qué opciones le corresponden
Monte Carlo	Evaluar casos muestreando datos o parámetros; indicar distribuciones, número de casos y semilla aleatoria, si corresponde.

Las ecuaciones, ramas, funciones y relaciones del método de integración pertenecen al MODELO; los valores suministrados para cada caso pertenecen a DATOS. USOS solicita las operaciones y sus opciones específicas. Las restricciones de una optimización son condiciones de esa búsqueda; no sustituyen las ecuaciones físicas del modelo. La semilla aleatoria de Monte Carlo tampoco se confunde con las SEMILLAS numéricas del modelo para resolver incógnitas.

La traducción debe conservar todos los usos y sus opciones aunque el motor elegido no pueda ejecutarlos. La falta de implementación de un análisis debe informarse explícitamente; no autoriza omitirlo ni cambiarlo por otro. Conservar una instrucción y ejecutarla son capacidades distintas.

2.1.3. Pedido POLOS_CEROS

La sintaxis acordada es un bloque `POLOS_CEROS` dentro de `USOS`, con una pareja `(salida / entrada)` por renglón. Se pueden escribir varias parejas; cada una solicita una transferencia escalar diferente. Los valores del punto de operación van en `DATOS`. `RESULTADOS` puede pedir otras magnitudes del punto, pero no elige la entrada ni la salida de la transferencia. La amplitud y la fase de una excitación AC tampoco forman parte de este pedido.

```
DATOS
  UFUENTE = 0
FIN
USOS
  POLOS_CEROS
    (IR12 / UFUENTE)
    (IL2 / UFUENTE)
  FIN
FIN
```

Regla	Significado
Entrada	Variable independiente del modelo que el análisis puede perturbar; las demás entradas y datos independientes permanecen fijos. En el ejemplo es `UFUENTE`.
Salida	Variable calculable a partir del modelo linealizado; en el ejemplo se piden dos transferencias independientes.
Punto de referencia	Solución estacionaria compatible con `DATOS`, con las derivadas de estado nulas. Si el punto no existe, no se determina o está en una discontinuidad, el pedido se rechaza.
Objeto calculado	Para cada pareja, `H(s)=`δ`salida(s)/`δ`entrada(s)` del sistema linealizado. Se informan polos y ceros finitos de esa transferencia, con multiplicidad; un modo interno cancelado exactamente no se presenta como polo de `H`.

Regla	Significado
Unidades	`s` es la variable de Laplace y tiene unidades inversas a `VARIABLE_INTEGRACION`; cuando esta es tiempo en segundos, los polos y ceros se informan en s^{-1}, con partes real e imaginaria.

El bloque solicita un análisis puntual de pequeña señal; no integra una trayectoria ni necesita una lista de frecuencias. `LINEALIZADO` en M puede pedir la preparación anticipada del modelo lineal a un motor que la necesite, pero por sí solo no solicita `POLOS_CEROS`. Siru/Calcula ya lo ejecuta sobre modelos diferenciales de hasta ocho estados con un punto estacionario determinado; el lector C conserva cada pareja y el resultado informa polos, ceros, multiplicidades y coeficientes de la transferencia. La ruta de convolución `K3` necesita su representación diferencial equivalente para este análisis: fijar los coeficientes instantáneos de memoria y linealizar ese punto no da la transferencia completa. Un motor que todavía no implemente el cálculo debe rechazar el pedido de forma explícita y conservarlo al traducir; no debe sustituirlo por una tabla AC.

2.1.4. Secuencia M, DU y R por entrada y salida estándar

Para una sesión continua, el flujo de entrada contiene un solo modelo M y luego uno o más DU. Cada documento lleva una envoltura exterior `M ... FIN` o `DU ... FIN`; los bloques interiores conservan sus propios `FIN`. Las envolturas permiten reconocer el final de cada DU sin esperar EOF. Un archivo M o DU usado aisladamente puede conservar la forma histórica sin envoltura y terminar en EOF.

```
M
  ...contenido de X.M.dsa...
FIN
DU
  ...datos y un pedido de simulación...
FIN
DU
  ...datos y un pedido POLOS_CEROS...
FIN
```

Entrada	Acción	Salida estándar
`M ... FIN`	Preparar y conservar el modelo para la sesión.	No produce un resultado de cálculo.
Primer `DU ... FIN`	Ejecutar ese pedido con sus propios datos, controles y resultados.	Emitir y vaciar un documento `R ... FIN` completo antes de procesar el próximo DU.
Cada DU siguiente	Reutilizar M, sin heredar datos ni controles del DU anterior, salvo una continuación pedida expresamente.	Emitir otro `R ... FIN` completo, en el mismo orden que los DU recibidos.

Cada R identifica el DU y contiene las secciones de resultado que ese DU pidió: por ejemplo, `SIMUL` para una trayectoria, o `PUNTO` con el análisis `POLOS_CEROS`. Si un DU contiene varios casos o usos, permanecen identificados y ordenados dentro de su R; no se mezclan con el R siguiente. Los diagnósticos y errores van a stderr. El resultado no se escribe a un archivo salvo que el

usuario redirija stdout o pida expresamente una copia. Siru `--dsa` ya procesa M y sucesivos DU puntuales o temporales, incluidos los análisis de punto de Calcula, con un R por DU antes de EOF. El punto se informa en líneas de texto codificadas dentro de `PUNTO`; falta definir una representación estructurada de sus variables y análisis.

Lo que el modelo declara para los usos (acuerdo del 28/10/2026). La medición de la primera fase de SCEPTRE (`sceptre2novender/PLAN-SEPARAR-MODELO-DATOS-USOS.md`, etapa 0) mostró que sensibilidad, optimización, peor caso y Monte Carlo comparten un mismo bloque, el adjunto, y que lo que lo determina no es cuál de los cuatro se pide sino qué variables intervienen: el generador inserta una fuente adjunta por cada variable observada. Por eso el MODELO declara dos listas y una palabra:

- `OBSERVABLES`: las variables respecto de las cuales se puede derivar (los dependientes de SCEPTRE). Si el modelo las declara, el generador produce además del modelo común un **modelo observable**: el mismo sistema con las fuentes adjuntas de esas variables, que sirve para sensibilidad, optimización, peor caso y Monte Carlo, y también para el transitorio solo.
- `VARIABLES`: lo que puede moverse (los independientes: parámetros, valores de elementos, con cotas o tolerancias). Si el modelo las declara, el adjunto se arma para ésas; si no, para todas las que el modelo admite.
- `LINEALIZADO`: pide el **modelo lineal** alrededor del punto de operación, el de AC, con fasores complejos y sus propias rutinas. Es otro modelo, no otro uso; necesita el punto de operación, que lo calcula el modelo común, y las fuentes AC del sistema son parte del modelo.

Así el generador produce hasta tres modelos de un mismo `.dsa`: el común, el observable y el lineal. Los USOS eligen después, en la segunda fase y sin volver a generar: qué conjuntos de sensibilidad entre los observables y variables declarados, qué objetivo y cotas, cuántas muestras, qué frecuencias. En la preparación de SCEPTRE, pedir un observable o una variable no declarados es un error de uso, no una regeneración silenciosa del modelo.

Alcance por motor. Las tres declaraciones anteriores responden a la preparación de SCEPTRE descrita en `sceptre2novender/PLAN-SEPARAR-MODELO-DATOS-USOS.md`. Para Calcula y el llamador de puntos de Siru, `OBSERVABLES`, la lista `VARIABLES` de análisis y `LINEALIZADO` se aceptan y conservan como metadatos: no agregan ecuaciones ni obligan a preparar otro modelo y no restringen qué salidas y datos puede seleccionar el servicio de sensibilidad. La lista VARIABLES de análisis no declara variables físicas nuevas. Los observables son las salidas dependientes; las variables independientes son los datos respecto de los cuales se deriva.

Calcula obtiene la sensibilidad local de las ecuaciones residuales del modelo cargado. `SENSITIVITY` en USOS del DU elige los pares de salidas y datos; no cambia M ni su `.dat`. El llamador de Siru delega esa operación en Calcula. La presencia de LINEALIZADO no solicita por sí sola AC: el ejecutor de DU ahora lo admite cuando USOS lo pide, a partir de un punto estacionario. También ejecuta Monte Carlo, peor caso y optimización local acotada, reutilizando el modelo preparado. Los métodos, opciones y límites están en Escribir modelos para Calcula (`../../../../calcula/calcula-6.0/ESCRIBIR-MODELOS.md#monte-carlo-optimizaciÃşn-peor-caso-y-ac-desde-du`). En modelos coordinados se estabiliza primero el punto y después se calcula la sensibilidad con la configuración de llaves fija: las ecuaciones de decisión se sustituyen por sus valores estables y no se vuelven a decidir llaves durante la diferenciación. Se conservan las ecuaciones físicas y las de lazos y cortes. La derivada corresponde a los datos del punto ya equilibrado, no a los anteriores al trasvase ni al salto de una conmutación. Si variar un dato con los demás estados finales fijos viola las restricciones del equilibrio, el pedido se rechaza: haría falta formular las variaciones dependientes de esos estados.

Nombre acordado: se usa `USOS` para esta parte del archivo. Reemplaza el encabezado `OPERACION` de los ejemplos anteriores. El traductor experimental escribe `USOS` y acepta

`OPERACION` como alias para los archivos anteriores. Su lector usa Slex (`dsa/dsa.slex`); el analizador consume sus tokens y capturas. No implementa todavía todos estos análisis: migrar el lector no implica completar la traducción ni la ejecución de cada uso. La sintaxis detallada de las opciones de cada uso se deberá concretar sin perder información del formato de origen.

2.2. 2. Contenido del modelo y origen de sus variables

Parte	Contenido
Variables (generadas automáticamente; no son una sección)	El programa obtiene las variables al procesar las ecuaciones, los elementos y conexiones, las derivadas, los métodos de integración y el tratamiento de lazos, cortes y cambios de conexión. El autor no las incorpora mediante una lista ni una sección de variables.
Ecuaciones	Las explícitas, las relaciones de derivadas, las aportadas por elementos y conexiones, las de métodos de integración y las generadas para lazos, cortes y cambios de conexión.
Parámetros	Variables incluidas en PARAMETROS: su valor es constante durante cada caso y sus derivadas valen cero. El valor puede suministrarse en el modelo o en los DATOS y puede variar entre casos.
Resultados solicitados (fuera de MODELO)	Selección para cada uso de variables de las que queremos ser informados, incluidos los parámetros, con unidades de salida opcionales. No define el modelo.
Semillas	Aproximaciones iniciales para las incógnitas, incluidas en el modelo para facilitar el cálculo; no fijan valores de la solución.
Funciones	Relaciones utilizadas por el modelo; las tablas son una forma particular de función y se procesan mediante Tabular.
Modelos de elementos	Relaciones reutilizables que incorporan las ramas y los elementos conectados.
Estados	Variables identificadas como estados propios del modelo general, además de los estados incorporados por los elementos.

No hay declaración separada de variables ni bloque VARIABLES. Las variables proceden de las fuentes enumeradas en la tabla, que incluye todas las posibles, intervengan o no en un caso particular. PARAMETROS, ESTADOS, SEMILLAS y RESULTADOS clasifican o utilizan esas variables; no las crean.

Lista opcional por ecuación (30/10/2026). En `ECUACIONES`, se admite `Z+T+VC3 = 0 V:Z,T,VC3`, también sin `= 0`. Sin `V:`, el lector obtiene los nombres de la expresión y los comunica al constructor del modelo. Con `V:`, utiliza la lista suministrada, sin ampliarla automáticamente; es una alternativa explícita si la detección no alcanza. No se añade una sección general de declaración de variables. La lista no corrige ecuaciones inválidas ni modifica la separación entre modelo, datos y usos.

El lector avisará si SEMILLAS, RESULTADOS o PARAMETROS contienen nombres que no corresponden a variables del modelo, indicando el nombre y su ubicación. La comprobación se

realiza después de obtener todas las variables, incluidas las generadas por elementos, conexiones y métodos de cálculo. Esas listas no crean variables nuevas.

Las ecuaciones del modelo se escriben como residuos igualados a cero. Se admite tanto `V - resistencia * I = 0` como `V - resistencia * I`: son equivalentes. Escribir `= 0` aporta claridad, pero es opcional. No son asignaciones ni indican qué variable debe despejarse. Esta decisión reemplaza la propuesta anterior de escribir una variable a la izquierda. Los datos y las definiciones de funciones conservan su notación propia: no son ecuaciones residuales del sistema.

2.3. 3. Nombres, parámetros y derivadas

Los prefijos especiales se reconocen en mayúscula. Las minúsculas no están reservadas: son válidos nombres como `tiempo`, `resistencia`, `potencia` y `capacidad`, sin anteponer una N artificial.

Prefijo	Significado acordado
`V`	Voltaje de una rama o diferencia de potencial entre terminales locales. No implica constancia.
`I`	Corriente de una rama o de un terminal local. No implica constancia.
`D`	Derivada de la variable cuyo nombre sigue, respecto de la variable de integración.
`R`	Función (relación).
`F`	Flujo; su significado físico preciso lo establece el modelo.
`T`	Tabla, entendida como un caso particular de función.
`E`	Variable de estado definida expresamente por el autor. Los estados generados por elementos no necesitan ese prefijo.
`N` u otra letra no reservada	Variable de otro tipo.

Se acuerda R para funciones (relaciones) y F para flujo, reemplazando el uso anterior de F para funciones. La correspondencia es Q con F y capacidad con inductancia.

El prefijo identifica el papel del nombre, no proporciona su valor ni determina por sí solo si ese valor cambiará. Los valores pueden estar fijados en el modelo, proporcionarse como datos o resultar de las ecuaciones.

La lista `PARAMETROS` clasifica variables como parámetros; no es una fuente de variables ni exige asignarles valor allí. Sus derivadas, incluidas las sucesivas, valen cero. Se elimina P como prefijo reservado: el carácter de parámetro lo determina la lista, no el nombre. Por ejemplo, incluir `capacidad` en PARAMETROS implica `Dcapacidad = 0`.

2.3.1. Variable de integración y derivadas

```
VARIABLE_INTEGRACION: tiempo
```

Con esa declaración, `DX` significa dX/dtiempo y `DF` significa dflujo/dtiempo. No hace falta escribir un bloque `DERIVADAS` ni repetir `DX = derivada(X, tiempo)`.

La presencia de derivadas no solicita dinámica. Por ejemplo, `DX + k * X = 0` con k=2 y X=3 permite calcular DX=−6 en un solo punto. Para una trayectoria debe pedirse explícitamente evolución durante un intervalo y aportar los datos iniciales y controles correspondientes. Un tiempo de evaluación aislado no solicita una trayectoria.

2.4. 4. Estados y dimensión del modelo

El estado del modelo se describe mediante un conjunto suficiente de variables independientes que, junto con los parámetros, las entradas y las ecuaciones, permite determinar su evolución. **La cantidad de variables de ese conjunto es la dimensión del espacio de estados.** Esta dimensión no es el número total de variables ni de incógnitas utilizadas por el programa.

Concepto	Significado
Variables de almacenamiento de los elementos	Magnitudes locales como la carga de un condensador o el flujo de una bobina. Al conectar los elementos pueden quedar relacionadas por restricciones.
Variables de estado independientes del modelo completo	Coordenadas independientes que describen el estado una vez consideradas todas las restricciones. Su cantidad determina la dimensión del espacio de estados.
Variables algebraicas	Magnitudes que se obtienen mediante relaciones del modelo y que no añaden por sí solas grados de libertad dinámicos.
Variables auxiliares del cálculo	Etapas del integrador y otras magnitudes introducidas para resolver; su cantidad no determina la dimensión del estado físico.

Por ejemplo, dos condensadores de capacidades constantes y positivas conectados en paralelo cumplen `Q1/capacidad1 - Q2/capacidad2 = 0`. Sus dos cargas no son independientes: pueden describirse mediante una sola variable, como el voltaje común. Sin otras restricciones, este conjunto aporta una dimensión de estado. Si una fuente ideal impone ese voltaje, deja de aportar un estado independiente.

En un modelo finito, con n variables de almacenamiento y r restricciones independientes regulares, el espacio admisible tiene dimensión local n−r. Deben considerarse todas las restricciones relevantes, no solo contar ecuaciones escritas. La dimensión puede cambiar con las conexiones o en situaciones singulares. Una memoria distribuida, como una convolución general, no necesariamente admite una representación exacta con un número finito de estados.

Una derivada también puede ser variable de estado. Ser derivada expresa una relación matemática con otra variable; ser estado identifica su papel en la descripción de la evolución. No son categorías excluyentes. En una bobina con corriente I y carga transportada Q relacionadas por `DQ - I = 0`, se puede usar I o, equivalentemente, DQ como variable de estado. No se cuentan ambas como estados independientes. La relación `V - inductancia * DI = 0` pasa a relacionar V con la segunda derivada de Q si se elige esa representación. Las derivadas sucesivas se escriben repitiendo D: DX, DDX, DDDX, DDDDX. Cada D inicial añade una derivación respecto de la variable de integración; esa secuencia se reserva y no forma parte de nombres base. Una derivada puede ser estado.

Uso	Qué implica identificar X como estado
Simulación temporal	El integrador utiliza su valor inicial y sus derivadas para calcular valores sucesivos compatibles con las ecuaciones.
Calcula: un punto	No ordena avanzar en el tiempo. X puede ser dato o incógnita según los datos y la operación; se resuelven las ecuaciones del punto.

Uso	Qué implica identificar X como estado
Búsqueda de equilibrio estacionario	La operación puede imponer derivadas de estado iguales a cero. Identificar un estado no impone esa condición automáticamente.

```
ESTADOS: X

DX + k * X = 0
```

Con X=3 y k=2, un cálculo puntual obtiene DX=−6. En una simulación se utiliza esa derivada para evolucionar X; para buscar un equilibrio estacionario se impone además DX=0.

La ausencia de una derivada no convierte una variable en parámetro. Por ejemplo, en `V - resistencia * I = 0`, V e I pueden variar entre instantes como variables algebraicas, aunque DV y DI no aparezcan. Tampoco se deduce DX=0 simplemente porque DX no aparezca. La constancia se establece mediante PARAMETROS o las relaciones correspondientes del modelo.

`ESTADOS` puede aparecer dentro de un modelo de elemento, para identificar sus estados locales, y en el modelo general, para identificar estados propios de sus ecuaciones fuera de los elementos. Al instanciar un condensador C1, su estado Q se incorpora como QC1; no hace falta repetirlo en la lista general. Estas listas identifican el papel de variables del modelo; no las crean. Al reunir estados locales y propios, el generador debe considerar sus restricciones para obtener el conjunto independiente del modelo completo. El número de nombres reunidos en ESTADOS no es automáticamente la dimensión.

2.4.1. Identificación explícita de estados del elemento

Dentro del modelo, `ESTADOS: Q` identifica Q como variable de estado local. Si hay varias, se separan por comas: `ESTADOS: I1, I3`. La declaración no asigna valores iniciales ni agrega una ecuación; esos valores se suministran como datos y deben ser compatibles con las conexiones. Tampoco obliga a calcular una trayectoria.

Modelo	Declaración	Relación con capacidad y flujo
Condensador con carga	`ESTADOS: Q`	`Q - capacidad * V = 0`; capacidad no es automáticamente un estado.
Condensador incremental	`ESTADOS: V`	Su capacidad incremental relaciona I con DV.
Bobina expresada mediante corriente	`ESTADOS: I`	La ley utiliza DI.
Bobina expresada mediante flujo concatenado	`ESTADOS: F`	La ley utiliza DF; para inductancia constante puede escribirse `F - inductancia * I = 0`.
Transformador con núcleo	`ESTADOS: F`	El flujo del núcleo es común a sus devanados; las tensiones incluyen el número de vueltas.

`capacidad` puede ser un dato, una magnitud calculada o una función de otras variables. Solo se declararía como estado si el modelo le atribuyera evolución propia y las ecuaciones

correspondientes. No debe confundirse capacidad con carga Q. En la bobina, el flujo concatenado y el flujo del núcleo del transformador tienen significados distintos, indicados por sus ecuaciones.

Los estados locales se particularizan igual que las demás variables: Q pasa a QC1 y F a FL1. No necesitan el prefijo E. Las conexiones pueden imponer restricciones entre ellos: declararlos no afirma que sean todos independientes en el sistema completo.

Un elemento algebraico sin memoria propia, como la resistencia ideal, no necesita declarar estados. Los elementos compuestos incorporan los estados de sus ramas internas; sus nombres incorporan recursivamente los contenedores, con separadores. La memoria de una convolución requiere una historia y no queda resuelta automáticamente con una lista finita de estados.

La correspondencia entre almacenamiento eléctrico y magnético es **Q con F** y **capacidad con inductancia**. El prefijo **R identifica funciones (relaciones)**; reemplaza el anterior uso de F para funciones. Por ejemplo, `Rflujo(I)` es una función y `F` es la variable de flujo. Los nombres de modelos históricos citados, como FD300, conservan su escritura original.

2.5. 5. Números, unidades y resultados

No habrá una sección UNIDADES. Cada número puede llevar su unidad allí donde aparezca: datos, constantes, semillas, ecuaciones, tablas o controles.

```
VB1 = 12 V
resistenciaR1 = 6 ohm
capacidadC1 = 100 uF
tiempo = 5 ms
```

Sin unidad explícita, se interpreta MKS/SI según la magnitud correspondiente. Las unidades explícitas se convierten al sistema interno. No se impone que toda variable tenga dimensión: un número puede ser adimensional.

Las unidades admitidas y sus conversiones son las que maneja el programa `units`; no se define un sistema propio de unidades para dsa.

2.5.1. Unidades solicitadas en resultados

En RESULTADOS se escriben variables solas o pares variable–unidad, **separados por comas**:

```
RESULTADOS
  tiempo ms, VC1 V, IC1 mA, QC1 uC, potencia
FIN
```

Una unidad junto a la variable indica la unidad de salida solicitada. Sin unidad explícita se informa en MKS/SI. Esto no cambia la ecuación ni el valor físico. Una unidad incompatible debe producir error. Las unidades compuestas se interpretan mediante `units`.

La magnitud es del modelo; la unidad es una elección (19/11/2026). Son dos cosas distintas y se decidían juntas por error. La *magnitud* de una variable –temperatura, flujo de calor, tensión, corriente– la determina el modelo del elemento al que pertenece, y en un diagrama la decide la hoja de formas con la que se dibujó: una resistencia conductiva de la hoja térmica y una resistencia de la hoja eléctrica no son el mismo modelo, aunque sus ecuaciones tengan la misma forma. La *unidad* en que se informa es una elección de quien pide el resultado, y para eso están la unidad junto a la variable en RESULTADOS y el bloque UNIDADES.

Lo térmico y lo eléctrico son **los dos** MKS/SI: el grado Celsius y el vatio son unidades SI igual que el volt y el ampere. Por eso no alcanza con "pasar a SI": si se pierde la magnitud, una temperatura se informa como una tensión en unidades base –`kg m^2 / A s^3`– y el resultado es correcto en SI pero no es lo que se modeló. Un circuito térmico escrito con modelos de elemento eléctricos pierde su dominio y nadie lo puede recuperar después: el dominio tiene que viajar en el modelo.

El cálculo es eléctrico; la equivalencia vive en los bordes (20/11/2026). Decisión del autor: **el simulador trabaja todo en variables eléctricas** –la `V`, la `I` y la `Q` de cada rama– y no se le pide que entienda dominios. Hacerlo trabajar con variables distintas según el dominio es mucho trabajo y no aporta: las ecuaciones son las mismas. Lo que sí hace falta es **decir cuáles son las equivalencias**, y que en lo que el autor ve aparezcan las variables equivalentes: los informes, los resultados, los circuitos y el DU.

La equivalencia es **una por dominio**, no una declaración por variable:

variable de cálculo	en lo eléctrico	en lo térmico	se escribe
`V` de una rama	tensión	temperatura	`T`
`I` de una rama	corriente	flujo de calor	`FLUJOQ`
`Q` de una rama	carga	calor acumulado	`Q`

El nombre del flujo de calor es `FLUJOQ` (decisión del autor, 20/11/2026). La documentación histórica de simplot ya traía el mecanismo para nombrarlo, con un alias: `J : W, flujo_de_calor,FLUJO`.

Una rama se declara por su nombre y su variable es otra. El modelo nombra las variables de todas las ramas igual, con la letra más el nombre de la rama y sin mirar el dominio, que es lo que corresponde si el cálculo es eléctrico: una fuente de flujo llamada `J1` tiene su variable en `IJ1`, la `I` de la rama `J1`. Quien declare la magnitud de una rama tiene que declararla también en esa forma, o la columna sale con la magnitud equivocada.

El dominio lo decide la **hoja de formas** con la que se dibujó, que el traductor conoce por objeto; de ahí sale la magnitud de cada variable sin que el autor declare nada. La unidad en que se informa sigue siendo una elección suya, y se dice donde se dicen las unidades.

2.6. 6. Métodos de integración

El método de integración es parte del modelo del sistema. Sus relaciones —RK, GLM u otras— generan ecuaciones y, cuando corresponde, variables de etapas e historia. Seleccionar un método no solicita por sí solo una evolución temporal.

La operación determina si se resuelve un punto o se calcula una trayectoria. Un modelo con derivadas puede utilizarse en un punto. Una trayectoria requiere los datos iniciales, intervalo y controles numéricos apropiados.

Para un método con memoria, continuar puede requerir más que el último estado. La definición del método incluye su arranque y manejo de memoria; los valores de la historia inicial son datos. No se confunden con semillas del solver.

2.7. 7. Modelos de elementos: relaciones y nombres locales

Son conceptos distintos y deben nombrarse sin confundirlos.

Concepto	Significado
Modelo del sistema	El problema completo que se entrega para una operación.
Modelo de elemento	Una definición reutilizable de las relaciones de un tipo de elemento.
Elemento concreto / rama	Una aplicación del modelo de elemento, con nombre y conexiones propias. Para dos terminales se habla de rama; también se admiten elementos de N terminales.

Las ecuaciones propuestas se escriben como **residuos iguales a cero, con `= 0` opcional**. `V - resistencia * I` y `V - resistencia * I = 0` son equivalentes; escribir el cero aporta claridad, pero no es obligatorio. Por ejemplo, `V - resistencia * I = 0` relaciona V, I y resistencia sin ordenar cuál debe calcularse. Esta decisión reemplaza la escritura anterior con una variable despejada. Las definiciones de funciones, como `Rrespuesta(i) = ...`, siguen definiendo funciones; no son restricciones residuales independientes. Las citas de las bibliotecas actuales conservan su sintaxis original.

Un modelo de elemento sencillo es un conjunto de ecuaciones con nombre. No hay un bloque de declaración `VARIABLES`; tampoco se requieren bloques redundantes `ARGUMENTOS`, `TERMINALES` o `ECUACIONES`.

```
MODELO_ELEMENTO Resistor
  V - resistencia * I = 0
FIN
```

Las funciones y tablas que aparecen en las ecuaciones se resuelven por sus definiciones. También pueden escribirse debajo del modelo uno o varios bloques de funciones locales; no es obligatorio repetirlas si ya están disponibles.

2.7.1. Nombres internos y externos

Cualquier variable local puede aparecer en un modelo de elemento. Al aplicarlo, su nombre se completa con el nombre del elemento:

Dentro del modelo	En el elemento C1
V	VC1
I	IC1
Q	QC1
DQ	DQC1
capacidad	capacidadC1

Así, dos capacitores tienen cargas diferentes, mientras que dos devanados dentro del mismo modelo de transformador comparten la variable local `F`. No se necesita introducir vínculos entre dos elementos para representar ese transformador: puede ser un único elemento de cuatro terminales.

Los nombres anidados se forman incorporando recursivamente el nombre del contenedor y preservando separadores. No se debe sustituir este acuerdo por una notación de propiedades como `R1.tension`: se conserva V e I como prefijos externos, por ejemplo VR1 e IR1.

2.8. 8. Circuito, terminales y nombres de instancias

CIRCUITO es una lista de ramas o elementos conectados. Una declaración identifica nombre de elemento, tipo y nodos en orden. La forma propuesta es:

```
CIRCUITO
  B1: FuenteTension, 1, 0
  R1: Resistor, 1, 0
FIN
```

Los nombres o números de los nodos se dan allí, no en el modelo de elemento. Un nombre como `positivo` sería solo un identificador de nodo, no una propiedad; los ejemplos usarán números cuando eso sea más claro. El uso de 0 como referencia se propuso, pero no se agrega una declaración redundante de nodos.

2.8.1. Índices y nombres de elementos anidados

Los índices se separan para evitar ambigüedades: `V1_12` corresponde a los terminales e1 y e12, mientras que `V11_2` corresponde a e11 y e2. Las abreviaturas de pares de un dígito usadas en los ejemplos, como V12, conservan su lectura e1–e2; cuando la escritura compacta sea ambigua se usan separadores.

Al expandir un modelo compuesto, cada variable interna incorpora el nombre del elemento contenedor, recursivamente y con separadores que preservan los límites de los nombres. Por ejemplo, `V_M1_A` identifica V de la rama A dentro de M1; `V_M2_A` pertenece a otra instancia. Si hay más niveles, `V_M1_B_A` conserva el camino de contenedores. La misma regla mantiene separados los nodos internos y las referencias locales al expandir las instancias. Para elementos simples se conservan los nombres ya utilizados, como VC1 y QC1.

2.8.2. Terminales y signos

Cada elemento debe ofrecer una corriente por terminal externo (`I1`, `I2`, ...), positiva entrando, y permitir calcular las diferencias de potencial entre todos ellos. No todas esas tensiones son independientes: por ejemplo, `V13 - V12 - V23 = 0`. En dos terminales se conserva la abreviatura V para V12 e I para I1; sin acumulación neta de carga, `I2 + I = 0`. Para varios terminales sin acumulación neta, la suma de las corrientes entrantes es cero. Si el modelo almacena carga neta total, el balance debe incluir su derivada. El generador debe evitar duplicar balances o identidades ya representados por ramas internas.

Nombre local	Interpretación
V, I	Convención de rama de dos terminales: V es potencial(a)−potencial(b); I es positiva de a hacia b.
I1, I2, ...	Corrientes positivas entrando al elemento por los terminales externos e1, e2, ...
V12, V23, ...	Diferencia de potencial entre los terminales locales indicados, en ese orden.

La cantidad de terminales se deduce de las variables y ecuaciones del elemento, incluidas las generadas por su circuito interno. No hay una sección que los declare. En `Q1: Transistor, 1, 2, 0`, los terminales e1, e2 y e3 se conectan a los nodos del circuito 1, 2 y 0. El orden de los nodos concreta la orientación en el circuito; no altera las ecuaciones locales.

Las tensiones cumplen relaciones de compatibilidad como V13=V12+V23 y V21=−V12. La interpretación y generación deben evitar añadir ecuaciones redundantes.

La presencia de circuito genera variables y ecuaciones adicionales. Cada rama aplica el modelo de su tipo; las conexiones generan relaciones entre ramas, como conservación de corrientes y diferencias de potencial. No se repiten manualmente todas esas variables y ecuaciones. La presencia de circuito no obliga a pedir evolución temporal.

2.8.3. Valor indicado en la rama

Se admite indicar directamente el valor al escribir la rama, como en el formato actual. Para una batería ideal, el valor corresponde a su tensión, sin crear un nombre intermedio como entrada o PE:

```
B1: Bateria, 1, 0 = 12 V
```

La orientación de la tensión debe respetar la convención del modelo y el orden de los nodos. La traducción de fuentes históricas debe preservar sus signos; no basta copiar el valor sin revisar

las convenciones. Esta regla queda acordada; su codificación en el traductor debe contrastarse con los casos originales.

2.9. 9. Composición mediante ramas internas

Un modelo de elemento puede incluir un **circuito interno**, además de ecuaciones explícitas y funciones. No queda limitado a una ecuación constitutiva única ni a dos terminales. Se admite descripción recursiva mediante otros modelos de elementos.

Las ramas internas generan sus propias variables y ecuaciones, que forman parte del modelo del elemento compuesto. Al usarlo dentro del sistema, deben conservarse las identidades internas sin colisiones entre distintas aplicaciones. Los terminales externos se deducen de la descripción; no se añade una lista obligatoria de terminales.

Dentro de la definición del elemento, **`e1, e2, ...` identifican terminales externos; los demás identificadores de nodo son internos**, incluidos los números sin el prefijo e. No hace falta una declaración separada de nodos. `I1` corresponde a e1 y `V12` es el potencial de e1 menos el de e2. Esta convención se aplica a los identificadores de conexión, no a los nombres de variables en general.

Ejemplo con dos terminales externos y un nodo interno:

```
MODELO_ELEMENTO ResistorCompuesto
  CIRCUITO
    A: Resistor, e1, medio
    B: Resistor, medio, e2
  FIN
FIN
```

Al usar `R1: ResistorCompuesto, 8, 3`, e1 se conecta con el nodo 8 y e2 con el nodo 3. El nodo `medio` es interno a R1; otra instancia tiene su propio nodo `medio`. Los nodos del circuito donde se usa el elemento pueden tener nombres o números.

Esta regla reemplaza la propuesta «números externos, nombres internos». Los nombres anidados incorporan recursivamente los nombres de sus contenedores, con separadores. La composición debe terminar en definiciones resolubles, no en una expansión circular infinita.

2.10. 10. Funciones y tablas locales

Un elemento puede usar varias funciones, incluidas funciones por tramos y tablas. Basta nombrarlas en sus ecuaciones si sus definiciones ya están disponibles. También pueden colocarse debajo de cada modelo los bloques de funciones que se necesiten; no se exige una sección de argumentos redundante.

Por ejemplo, `V - (Rresistencia(I)) = 0` expresa una relación constitutiva, mientras que `V - (Tcaracteristica(I)) = 0` usa una característica tabulada. Todas las tablas se procesan mediante Tabular, que proporciona su evaluación e interpolación; dsa no implementa otro mecanismo. Falta precisar la escritura y el alcance de los bloques y cómo se expresan las opciones de Tabular.

2.10.1. Funciones dentro de un modelo: ejemplo completo

En el modelo actual FD300 de `modellib.siru`, la capacidad de una rama depende de dos funciones locales. Extracto literal de esas definiciones:

```
FUNCTIONS
  Q1J (A,B,C,D) = (A/(B-C)**D)
  Q1D (E,F,G) = (E*(F+G))
```

El siguiente ejemplo muestra una propuesta de elemento de dos terminales con **dos funciones definidas dentro del modelo**, después de sus ecuaciones. Reproduce la forma funcional de una capacidad de juntura, no pretende convertir por sí solo todo FD300:

```
MODELO_ELEMENTO CondensadorDeJuntura
  LOOP
  ESTADOS: V
  I - (Rcapacidad(V) * DV) = 0

  FUNCIONES
    Rcapacidad(v) = capacidadBase * Rfactor(v)
    Rfactor(v) = (1 - v / tensionBarrera)^(-exponente)
  FIN
FIN
```

Parte	Interpretación
`Rcapacidad(V)`	Capacidad incremental dQ/dV evaluada en el voltaje de la rama.
`Rfactor(v)`	Segunda función local, llamada por la primera.
`v`	Argumento local de la función; recibe el valor de V.
`capacidadBase`, `tensionBarrera`, `exponente`	Magnitudes locales del elemento, cuyos valores pueden suministrarse al usarlo.
Dominio del ejemplo	`tensionBarrera > 0` y `V < tensionBarrera`; fuera de él se necesita otra ley.

`FUNCIONES ... FIN` es una escritura ilustrativa, no una gramática aprobada. El acuerdo es que puede haber varias funciones locales debajo de las ecuaciones. En particular, falta precisar cómo las funciones acceden a magnitudes locales como `capacidadBase`, para que cada instancia conserve sus propios valores.

2.10.2. TABLE y opciones de Tabular

Toda función tabulada se procesa mediante Tabular. En la interfaz existente se escribe:

```
Table('archivo.txt', 'X', 'Y', valor, metodo)
```

Argumento u opción	Significado
archivo.txt	Archivo o nombre de una tabla registrada en memoria.
X	Columna utilizada como argumento de la función.
Y	Columna del valor de la función.
valor	Punto donde se evalúa.
metodo = 1	Interpolación lineal.
metodo = 2	Interpolación spline.

En los ejemplos de tablas locales, la opción de método se coloca junto al encabezado: `TABLE Tcorriente(v), metodo=1` o `TABLE Tfactor(i), metodo=2`. Esta es la escritura propuesta

para dsa; los valores 1 y 2 son los admitidos por Tabular. No se crea un interpolador nuevo. La extrapolación y el tratamiento de saltos se conservan según Tabular; la interfaz citada no ofrece un argumento adicional para seleccionar otra política de extrapolación.

2.10.3. Tablas dentro de un modelo: ejemplo completo

El modelo actual `2N9999AA` incluye esta tabla local en `FUNCTIONS`:

```
DIODE TABLE 1
  0,0,.3,0,.65,.05,.7,.6,.72,1.4,.73,2,.74,3.4,.77,10,.8,22
```

Son pares de valores de la tabla histórica. Para convertirla deben verificarse sus unidades y la regla de interpolación del formato de origen; no se les asignan unidades nuevas por suposición.

Este ejemplo nuevo de diodo tabulado muestra explícitamente las unidades y la tabla **dentro del modelo**. Sus números son didácticos, no datos de un dispositivo comercial:

```
MODELO_ELEMENTO DiodoTabulado
  I - (Tcorriente(V)) = 0

  TABLE Tcorriente(v), metodo=1
    0 V,    0 A
    0.5 V,  0.1 mA
    0.6 V,  1 mA
    0.7 V,  10 mA
  FIN
FIN
```

La primera columna es el argumento y la segunda el valor de la función tabulada. El encabezado indica metodo=1 (lineal); puede elegirse metodo=2 (spline). Se conserva la extrapolación de Tabular: el ejemplo no impone una política adicional de rechazo fuera del intervalo. TABLE ... FIN es la escritura propuesta para alojar la tabla dentro del modelo.

2.10.4. Función que usa una tabla local

Una tabla puede intervenir dentro de una función del mismo modelo, en lugar de ser la ecuación constitutiva completa:

```
MODELO_ELEMENTO ResistenciaTabulada
  V - (Rrespuesta(I)) = 0

  FUNCIONES
    Rrespuesta(i) = resistenciaBase * i * Tfactor(i)
  FIN

  TABLE Tfactor(i), metodo=1
    0 A,  1
    1 A,  1.1
    2 A,  1.4
  FIN
FIN
```

Aquí los valores de Tfactor son adimensionales. El ejemplo solo cubre corrientes entre 0 y 2 A; no presupone una extensión simétrica a corrientes negativas. Ilustra que una tabla es una función y que las definiciones pueden coexistir dentro del elemento, sin convertirlas en recursos obligatoriamente externos.

2.11. 11. Catálogo de ejemplos de elementos

Los siguientes ejemplos registran las familias discutidas. Son propuestas de notación, no una biblioteca implementada ni una conversión certificada de todo el corpus. Los nombres locales que no sean funciones se particularizan para cada aplicación del modelo.

```
MODELO_ELEMENTO Resistencia
  V - resistencia * I = 0
FIN

MODELO_ELEMENTO ResistenciaNoLineal
  V - (Rresistencia(I)) = 0
FIN

MODELO_ELEMENTO CapacitorTotal
  LOOP
  ESTADOS: Q
  Q - capacidad * V = 0
  DQ - I = 0
FIN

MODELO_ELEMENTO CapacitorIncremental
  LOOP
  ESTADOS: V
  I - capacidad * DV = 0
FIN

MODELO_ELEMENTO Bobina
  CUTSET
  ESTADOS: I
  V - inductancia * DI = 0
FIN

MODELO_ELEMENTO BobinaFlujo
  CUTSET
  ESTADOS: F
  F - (Rflujo(I)) = 0
  V - DF = 0
FIN

MODELO_ELEMENTO BobinaNoLineal
  CUTSET
  ESTADOS: I
  inductancia * DI - V^exponente = 0
FIN

MODELO_ELEMENTO BateriaConCarga
  LOOP
  ESTADOS: Q
  DQ - I = 0
  V - (tensionLlena - coeficiente * (cargaMaxima - Q) / Q) = 0
FIN

MODELO_ELEMENTO FuenteTensionFuncional
  LOOP
  V - (Rtension(tiempo)) = 0
FIN

MODELO_ELEMENTO FuenteCorrienteFuncional
```

```
  CUTSET
  I - (Rcorriente(tiempo)) = 0
FIN

MODELO_ELEMENTO Diodo
  I - (corrienteSaturacion * (exp(V / tensionTermica) - 1)) = 0
FIN
```

Capacidad total Q/V y capacidad incremental dQ/dV son definiciones distintas cuando dependen del estado. La bobina no lineal del ejemplo conserva la ley concreta encontrada en el corpus, no representa toda bobina no lineal. La expresión de batería exige un dominio en que la división por Q sea válida.

Las fuentes funcionales no obligan a constancia. Para una fuente cuyo voltaje se proporciona directamente como VB1 en DATOS, sigue pendiente la escritura mínima de su modelo sin introducir una variable intermedia innecesaria ni una ecuación ficticia.

2.11.1. Elementos de cuatro terminales

```
MODELO_ELEMENTO Transformador
  ESTADOS: F
  V12 - vueltas1 * DF = 0
  V34 - vueltas2 * DF = 0
  I1 + I2 = 0
  I3 + I4 = 0
  vueltas1 * I1 + vueltas2 * I3 - reluctancia * F = 0
FIN

MODELO_ELEMENTO TransformadorIdeal
  V12 - relacion * V34 = 0
  I3 + relacion * I1 = 0
  I1 + I2 = 0
  I3 + I4 = 0
FIN

MODELO_ELEMENTO BobinasAcopladas
  ESTADOS: I1, I3
  V12 - (inductancia1 * DI1 + mutua * DI3) = 0
  V34 - (mutua * DI1 + inductancia2 * DI3) = 0
  I1 + I2 = 0
  I3 + I4 = 0
FIN

MODELO_ELEMENTO Girador
  V12 - coeficiente * I3 = 0
  V34 + coeficiente * I1 = 0
  I1 + I2 = 0
  I3 + I4 = 0
FIN
```

El transformador es un único elemento de cuatro terminales. Su flujo es una variable interna común: en TR1 pasa a FTR1 y su derivada a DFTR1. No requiere dividirlo en dos devanados externos ni introducir una sección VINCULOS. Las ecuaciones de BobinasAcopladas suponen inductancias y mutua constantes en el tiempo. Para coeficientes variables deben usarse las relaciones correspondientes, sin omitir sus derivadas.

2.12. 12. Lazos, cortes y responsabilidad del generador

El generador debe construir las ecuaciones adicionales que correspondan a los lazos y cortes del esquema. Los modelos de elementos aportan sus relaciones constitutivas y las marcas de vigilancia topológica; no contienen por separado la ecuación global de la configuración.

En particular, deben contemplarse los lazos de capacitores y los cortes de bobinas, incluyendo la presencia de fuentes y los cambios de conexión. En el código existente hay relaciones para la evolución compatible y para redistribuciones instantáneas de carga o flujo. El nuevo lenguaje debe preservar ese contenido.

2.12.1. Condensador: ecuaciones propias del elemento

```
MODELO_ELEMENTO Condensador
  LOOP
  ESTADOS: Q
  Q - capacidad * V = 0
  DQ - I = 0
FIN
```

Aquí `Q` es la carga de la placa del primer terminal, `V` es su potencial menos el del segundo terminal e `I` entra por el primero. La variable de integración es el tiempo. **`DQ` significa derivada de Q; su igualdad con I la establece este modelo, no el lenguaje.** Si Q representara otra magnitud, no se deduciría esa igualdad.

Para la instancia C1, las ecuaciones pasan a ser `QC1 - capacidadC1 * VC1 = 0` y `DQC1 - IC1 = 0`. No se da valor a la capacidad al definir el tipo.

Si la capacidad es constante, se obtiene `DV = DQ/capacidad`. Si depende de V, la relación que corresponde es `DQ - ((capacidad + V*d(capacidad)/dV)*DV) = 0`. Una dependencia de otras variables añade sus términos de derivación. Esta última escritura es matemática explicativa, no una gramática nueva acordada.

2.12.2. Ejemplo de condensadores en un corte (cutset)

Consideremos dos condensadores C1 y C2 en paralelo entre los nodos a y b, sin otras conexiones. Orientamos ambas ramas de a hacia b. El corte que separa a de b contiene las dos ramas.

Procedencia	Ecuación
Modelo de C1	`QC1 - capacidadC1 * VC1 = 0`, `DQC1 - IC1 = 0`
Modelo de C2	`QC2 - capacidadC2 * VC2 = 0`, `DQC2 - IC2 = 0`
Corte, conservación de corriente	`IC1 + IC2 = 0`
Corte, usando las ecuaciones de los condensadores	`DQC1 + DQC2 = 0`
Conservación de carga resultante	`QC1 + QC2 - cargaTotal = 0`

`cargaTotal` es la carga neta conservada del nodo a, determinada por el estado inicial. Si una fuente inyecta una corriente J en a y la retira de b, el corte incluye también esa rama: `DQC1 + DQC2 - J = 0`. La carga total deja de ser constante.

La ecuación del corte depende de la conexión y de sus orientaciones: no puede escribirse dentro de un condensador aislado nombrando otros elementos que aún no existen. El modelo

aporta `DQ - I = 0`; el generador combina las contribuciones al reconocer el corte. Las filas equivalentes no se agregan todas como restricciones independientes.

2.12.3. Ejemplo de lazo de condensadores y ecuación derivada

Los mismos dos condensadores en paralelo forman además un lazo. Para capacidades constantes:

Relación	Ecuación
Compatibilidad de voltajes	`VC1 - VC2 = 0`
Ley del lazo derivada respecto del tiempo	`DVC1 - DVC2 = 0`
Sustitución de las relaciones de los elementos	`DQC1/capacidadC1 - DQC2/capacidadC2 = 0`
Sustitución de DQ por I, autorizada por estos modelos	`IC1/capacidadC1 - IC2/capacidadC2 = 0`

Con una fuente de tensión en el recorrido, por ejemplo `VC1 + VC2 - VB1 = 0`, la ecuación derivada es `DQC1/capacidadC1 + DQC2/capacidadC2 - DVB1 = 0`. La derivada de la fuente solo es cero si su tensión es constante.

La ley derivada no reemplaza la compatibilidad inicial: conservar una diferencia de voltajes no garantiza que esa diferencia sea cero. El estado inicial debe satisfacer la ley original. Si una llave conecta condensadores con voltajes inicialmente incompatibles, el tratamiento ideal de redistribución requiere además conservación de carga y compatibilidad final; la corriente instantánea no queda descrita por una evolución suave ordinaria.

En el generador existente, la degeneración tratada especialmente es la de **lazos de condensadores** y, de forma dual, **cortes de bobinas**. Un corte con condensadores sí tiene su ley de corrientes, como muestra el ejemplo; no debe confundirse con ese caso especial de cortes de bobinas.

2.12.4. Qué pertenece al elemento y qué pertenece al esquema general

Las ecuaciones globales de lazo y corte pertenecen al esquema de conexiones, no a cada modelo de elemento aislado. El modelo aporta sus relaciones físicas; el generador las combina según la configuración y los signos de las conexiones.

Nivel	Responsabilidad	Ejemplo
Elemento	Relaciones constitutivas, estados y relaciones necesarias para obtener derivadas.	Condensador: `Q - capacidad * V = 0`, `DQ - I = 0`. Con capacidad constante se deduce `capacidad * DV - I = 0`.
Esquema general	Compatibilidad de tensiones y conservación de corrientes según las conexiones.	Lazo: suma orientada de V igual a cero. Corte: suma orientada de I igual a cero.
Generador	Detectar la configuración y construir o seleccionar las ecuaciones necesarias.	Ley derivada de lazo: suma orientada de DV igual a cero; ley derivada de corte: suma orientada de DI igual a cero.

Las relaciones derivadas de los elementos y las leyes globales no deben añadirse indiscriminadamente como ecuaciones independientes: algunas son consecuencias o reemplazos de otras en la formulación elegida.

2.12.5. Marcas LOOP y CUTSET

Se acuerdan las marcas `LOOP` y `CUTSET` como indicaciones de vigilancia topológica dentro del modelo. No contienen una ecuación global ni afirman que el elemento ya esté en un lazo o corte: eso depende de su conexión.

Familia de dos terminales	Marca	Relación que puede aportar al cálculo global
Condensador	`LOOP`	DV a partir de su ley de carga y corriente.
Fuente ideal de tensión, incluida batería ideal	`LOOP`	La derivada de su tensión; no se supone cero salvo que sea constante.
Bobina	`CUTSET`	DI a partir de su ley constitutiva.
Fuente ideal de corriente	`CUTSET`	La derivada de su corriente; no se supone cero salvo que sea constante.

Por ejemplo, con dos condensadores y una fuente en un lazo, la conexión puede dar `VC1 + VC2 - VB1 = 0`. Para capacidades constantes, su ecuación derivada es `IC1/capacidadC1 + IC2/capacidadC2 - DVB1 = 0`. Cada condensador aporta su relación; ninguno conoce por sí solo los nombres C2 o B1.

En un corte de dos bobinas y una fuente de corriente, una orientación posible da `IL1 + IL2 - IJ1 = 0`. Para inductancias constantes y las convenciones de rama correspondientes, la ley derivada es `VL1/inductanciaL1 + VL2/inductanciaL2 - DIJ1 = 0`. Los signos dependen del corte concreto.

Si cambia una conexión, se revisan los lazos y cortes de la nueva configuración. El ajuste del estado mediante conservación de carga o flujo y compatibilidad es distinto de las ecuaciones derivadas usadas para evolucionar después del cambio. Una ley derivada no establece por sí sola el estado compatible inicial ni representa un salto impulsivo como si fuera una derivada ordinaria finita.

Queda precisar cómo obtiene el generador las derivadas de funciones y tablas, qué debe aportar el autor cuando no puedan obtenerse, y cómo se aplica la vigilancia a elementos de varios terminales o a sus ramas internas. No falta introducir una ecuación global particular de lazo o corte dentro de cada modelo.

2.12.6. Condiciones matemáticas: lógica interna del programa

Por ahora, reconocer estos conjuntos y construir sus ecuaciones es responsabilidad del programa. No se incorporan secciones GRUPAL a los modelos. Se registra como posible extensión futura un mecanismo `GRUPAL nombre`, con condiciones de selección y reglas de actuación sobre los elementos seleccionados; no está implementado ni forma parte de la sintaxis actual de la propuesta.

Sea A la matriz de incidencia orientada del circuito: las filas representan nodos y las columnas ramas. Fijamos $+1$ en el origen de cada rama y -1 en su destino, con corriente positiva del origen al destino y tensión igual al potencial del origen menos el del destino. Las expresiones siguientes se aplican al grafo de ramas de dos terminales; el tratamiento de elementos de varios terminales requiere precisar su representación.

Conjunto	Condición matemática
LOOP	Un vector no nulo c, con componentes $0, +1, -1$, cumple $Ac = 0$. Su soporte identifica las ramas y sus signos fijan la orientación del recorrido. Para un lazo simple se exige soporte mínimo entre los vectores no nulos del espacio de ciclos.
CUTSET	Un vector de nodos s, con componentes $0, 1$, define una partición. El vector $b = A^{\mathsf{T}} s \neq 0$ identifica las ramas que cruzan la partición y sus orientaciones. Para un corte mínimo se exige soporte mínimo entre los cortes no vacíos.

La condición de elegibilidad se agrega a la condición topológica:

$$\text{soporte}(c) \subseteq E_{\text{LOOP}}, \qquad \text{soporte}(b) \subseteq E_{\text{CUTSET}}.$$

Los conjuntos E contienen las ramas elegibles según sus modelos: típicamente condensadores y fuentes ideales de tensión para LOOP, y bobinas y fuentes ideales de corriente para CUTSET. Las marcas de los elementos indican elegibilidad; no enumeran un conjunto concreto ni describen el algoritmo.

Con V e I como vectores de tensiones y corrientes de rama, las leyes y sus versiones derivadas son:

$$c^{\mathsf{T}} V = 0, \qquad c^{\mathsf{T}} \dot{V} = 0; \qquad b^{\mathsf{T}} I = 0, \qquad b^{\mathsf{T}} \dot{I} = 0.$$

La ley de corte anterior supone conservación de corriente en los nodos del grafo; si se representa almacenamiento neto de carga nodal, corresponde incluir su balance o representarlo mediante ramas apropiadas. Las leyes derivadas se aplican durante intervalos de configuración fija y evolución diferenciable, no directamente a un salto impulsivo al conmutar.

El programa detecta conjuntos elegibles, selecciona relaciones independientes y utiliza las ecuaciones originales o derivadas que requiera la formulación. No agrega todas como restricciones independientes. La compatibilidad inicial y el ajuste ante cambios de conexión se tratan por separado. Esta descripción fija el criterio matemático; no afirma que se haya implementado un lector dsa ni un mecanismo genérico GRUPAL.

2.13. 13. Llaves y cambios de configuración

Las llaves no son ramas comunes ni modelos de elementos. Se tratan en una **sección especial** porque cambian el circuito: al cerrar o abrir modifican las conexiones. Debe identificarse la conexión afectada y la condición de cambio.

Se conserva la escritura actual de las llaves y su mecanismo de conmutación; no se diseña una sintaxis nueva. Por ejemplo, `S1, 2-3 = T<1` en .d y `SB-S1, 2-3 = (T<1)+(T>3)` en .ds. El valor verdadero o 1 indica cerrada y 0 indica abierta. Su tratamiento sigue siendo especial porque modifica la configuración.

La propuesta admite calcular un punto con llaves: la configuración determina el circuito y sus ecuaciones. Si las condiciones evaluadas en la solución exigen otra configuración, se cambia y se vuelve a resolver, con los ajustes correspondientes, hasta obtener una configuración compatible. Si la configuración no se estabiliza, se informa «circuito imposible». Ese diagnóstico se refiere a la falta de una configuración estable en este procedimiento; no debe confundirse con un fallo numérico del solver.

2.13.1. Opción implementada: llaves resistivas (1/11/2026)

Dentro de MODELO se admite `LLAVES_RESISTIVAS`, con `GENERAL=[RON,ROFF]` y entradas por nombre de llave que prevalecen sobre GENERAL. Los valores son finitos, numéricos en ohmios y cumplen `0<RON<ROFF`. Sin opción, las llaves siguen siendo ideales. La opción queda en M porque cambia sus ecuaciones; la condición y la coordinación se conservan. Las ramas seleccionadas tienen resistencia finita tanto abiertas como cerradas y no producen el trasvase instantáneo ideal. El tratamiento de un punto con llaves ya está implementado en Calcula mediante el coordinador de Siru.

La guía de llaves resistivas (`../../../siru/doc/LLAVES-RESISTIVAS.md`) especifica la sintaxis en los tres formatos, las pruebas, la diferencia entre punto y trayectoria y la función permanente LLAVS para resistencias de SCEPTRE. En SCEPTRE esa función se evalúa cuando el motor evalúa la resistencia; no incorpora la coordinación entre pasos de Siru. La exportación automática a `.d` nativo de SCEPTRE convierte todas las llaves en resistencias con LLAVS. Exige RON/ROFF para cada llave, mediante GENERAL o entradas individuales; si faltan, enumera las llaves afectadas y termina con error 101 sin escribir la salida. No inventa valores. El traductor acepta `.d`, `.ds`, `.dsa` y M/DU separados. Los límites explícitos de conversión y las pruebas están en esa guía. La selección parcial y las llaves ideales por omisión siguen correspondiendo a Calcula/Siru.

2.14. 14. Elementos con memoria: convolución

Se propuso un operador matemático explícito para describir elementos con memoria:

```
MODELO_ELEMENTO ImpedanciaConMemoria
  V - (convolucion(Rrespuesta, I)) = 0
FIN

MODELO_ELEMENTO AdmitanciaConMemoria
  I - (convolucion(Rrespuesta, V)) = 0
FIN
```

La intención causal es integrar Rrespuesta(t−s) por la señal de entrada en s sobre la historia correspondiente. La respuesta puede estar definida mediante una función o una tabla.

La relación de convolución pertenece al modelo; la historia inicial pertenece a los datos. Evaluar un punto no elimina la necesidad de conocer la contribución de su historia. La parte instantánea, si existe, debe conservarse. Se conserva el tratamiento actual de la convolución, incluida su historia, inicialización, parte instantánea y evaluación numérica; no se propone un mecanismo nuevo.

2.15. 15. Ejemplo completo: un modelo, datos y un cálculo puntual

Este ejemplo reúne las partes del lenguaje sin circuito. Los encabezados MODELO, DATOS y USOS muestran la organización propuesta; no afirman que exista ya un lector que ejecute este texto. Los ejemplos anteriores muestran cómo se incorporan ramas y modelos de elementos.

```
MODELO
  VARIABLE_INTEGRACION: tiempo
  ESTADOS: X
  PARAMETROS: k

  ECUACIONES
    DX + k * X = 0
  FIN

  SEMILLAS
```

```
    DX = -1
  FIN

FIN

RESULTADOS
  X, DX, k
FIN

DATOS
  X = 3
  k = 2
FIN

USOS
  PUNTO
FIN
```

Parte	Efecto
Ecuación	Introduce X, DX y k, con DX derivada de X.
PARAMETROS	Clasifica k como constante en el caso; no crea k.
ESTADOS	Identifica X como estado; no solicita evolución.
SEMILLAS	Propone DX=−1 como arranque numérico; no fija la solución.
DATOS	Fija X=3 y k=2 para el caso.
USOS	Solicita un punto; la solución es DX=−6.
RESULTADOS	Informa X, DX y k; los parámetros también pueden ser resultados.

Otro bloque DATOS con X=4 y k=2 describe otro caso y produce DX=−8. No es por sí mismo el punto siguiente de una trayectoria. Para evolucionar se requiere la operación temporal y el método correspondiente. Las unidades se omiten aquí; se interpretan en MKS/SI según las magnitudes.

2.16. 16. Comparación con las bibliotecas actuales

Esta comparación toma como fuentes locales `slex/d2datdinc/biblioteca/libraryori.siru`, `slex/d2datdinc/biblioteca/modellib.siru` y la sección «Cómo resuelve cada caso» de `slex/din/README.md`. Distingue la biblioteca elemental, los modelos compuestos de SCEPTRE y las ecuaciones que añade el generador. No son una única sintaxis actual.

Aspecto	Actualmente	Propuesta dsa
Ecuaciones elementales	Expresiones residuales con cero implícito.	Residuos con `= 0` opcional; no asignan una variable ni fijan cuál es incógnita.
Valores del elemento	Posiciones `x1`, `x2`, etc.; el número entre paréntesis indica su cantidad.	Nombres locales significativos, como `capacidad` y `resistencia`.

Aspecto	Actualmente	Propuesta dsa
Magnitudes de rama	`VOLTAGE`, `CURRENT`, `CHARGE`.	`V`, `I`, `Q`, `F`, `capacidad`, `inductancia`, particularizadas por instancia.
Estado y derivada	Metadatos `IC: CHARGE:CHARGE:DCHARGE`, por ejemplo.	`ESTADOS: Q` identifica el estado; `DQ` expresa derivación y `DQ - I = 0` es una ecuación del condensador.
Funciones locales	Los modelos compuestos pueden incluir `FUNCTIONS`.	Se conservan definiciones locales junto a cada modelo, sin lista obligatoria de argumentos del elemento. Los argumentos matemáticos de cada función sí forman parte de su definición.
Tablas locales	Tablas en `FUNCTIONS`, usadas directamente o desde ecuaciones.	Funciones tabuladas locales con nombres T, procesadas mediante Tabular; falta cerrar su escritura y el paso de opciones.
Ramas internas	`MODEL ...`, seguido de `ELEMENTS` y conexiones.	Composición recursiva mediante modelos de elementos y ramas internas.
Terminales	Los modelos compuestos declaran su lista, por ejemplo `(B-E-C)`.	Se reconocen por variables y ecuaciones; en las ramas internas, `e1`, `e2`, `...` son externos y los demás identificadores de nodo son internos.
Lazos y cortes	El generador reconoce configuraciones y arma ecuaciones especiales.	Marcas LOOP/CUTSET en los modelos; el generador arma las leyes globales según las conexiones y usa las relaciones constitutivas.
Llaves	Hay una definición elemental `SB` y tratamiento de configuración.	Sección especial de cambios de conexión, fuera de los elementos ordinarios.

2.16.1. Resistencia: definición actual y propuesta

Actual, en `libraryori.siru`:

```
RB(1) = VOLTAGE - x1 * CURRENT
```

Propuesta:

```
MODELO_ELEMENTO Resistencia
  V - resistencia * I = 0
FIN
```

La relación física no cambia: el residuo actual es cero. `x1` pasa a tener el nombre local `resistencia`; para R1 se usa `resistenciaR1`. Ese nombre no obliga a un valor constante.

2.16.2. Condensador: ecuaciones y estado

Actual, en `libraryori.siru`:

```
CB(1) = IC: CHARGE:CHARGE:DCHARGE,
        DCHARGE - CURRENT V: , x1 * VOLTAGE - CHARGE
```

Propuesta:

```
MODELO_ELEMENTO Condensador
  LOOP
  ESTADOS: Q
  Q - capacidad * V = 0
  DQ - I = 0
FIN
```

Se conservan las dos ecuaciones. La igualdad entre corriente y derivada de carga no se incorpora como significado del prefijo D: se escribe dentro del modelo. El actual `IC` identifica el estado; en la propuesta esa selección queda explícita en `ESTADOS: Q`.

Para el condensador incremental actual:

```
CINC(1) = IC: VOLTAGE:VOLTAGE:DVOLTAGE,
          DVOLTAGE * x1 - CURRENT V:
```

La propuesta equivalente es:

```
MODELO_ELEMENTO CondensadorIncremental
  LOOP
  ESTADOS: V
  I - capacidadIncremental * DV = 0
FIN
```

Aquí la capacidad es dQ/dV, no Q/V. La biblioteca actual advierte que CINC no puede participar en su tratamiento de lazos de condensadores porque no tiene carga como estado. Cambiar la escritura no resuelve esa limitación: para admitirlo allí hay que definir la relación de carga y su tratamiento correspondiente.

2.16.3. Bobina y fuente: traducción de relaciones y signos

Actual en la biblioteca	Propuesta equivalente	Condición
`DCURRENT * x1 - VOLTAGE`	`inductancia * DI - V = 0`	Conserva la ley concreta de LB.
`DCURRENT * x1 - VOLTAGEˆx2`	`inductancia * DI - Vˆexponente = 0`	Conserva LNL; no es una ley universal de bobinas.
`CURRENT - x1`	`I - valor = 0`	Conserva la orientación de corriente de JB.
`VOLTAGE + x1` con `INV:VOLTAGE`	Revisar la orientación y la inversión al traducir.	No puede sustituirse ciegamente por `V - valor = 0`.

La biblioteca actual usa una convención especial para fuentes de tensión. Antes de convertir una instancia hay que verificar el orden de sus nodos y el tratamiento de `INV`; el nuevo nombre de la variable no garantiza por sí solo equivalencia de signos.

2.16.4. Modelos con ramas internas: qué se conserva

El modelo actual FD300 contiene cuatro ramas, un nodo interno y funciones propias:

```
MODEL FD300 (PERM) (1-3)
  ELEMENTS
    CD, 1-2 = Q1J(4.618,.9,VCD,.484)
              + Q1D(1.529E4,JD,8.2045E-8)
    RB, 2-3 = .000303
    RS, 1-2 = 6E8
    JD, 1-2 = DIODE Q(8.2045E-8,25.239)
  FUNCTIONS
    Q1J (A,B,C,D) = (A/(B-C)**D)
    Q1D (E,F,G) = (E*(F+G))
```

El salto de línea de CD se usa aquí solo para facilitar la lectura. Una traducción debe conservar esas cuatro ramas, la distinción entre los terminales externos 1 y 3 y el nodo interno 2, las dos funciones y las dependencias cruzadas. No basta reemplazarlo por la ecuación de un diodo ideal.

En dsa se propone incluir `CIRCUITO` y funciones locales dentro de `MODELO_ELEMENTO`, como en el ejemplo de composición recursiva de este documento. Para FD300, los antiguos terminales externos 1 y 3 se traducen a e1 y e2; el nodo 2 permanece interno. Así, CD, RS y JD conectan e1 con 2, y RB conecta 2 con e2. La distinción de terminales queda resuelta por el prefijo e. La conversión completa debe aplicar la composición de nombres y conservar las leyes y funciones particulares del dispositivo.

2.16.5. Lazos y cortes: dónde están hoy las ecuaciones adicionales

Caso	Tratamiento actual documentado en libdin	Qué debe expresar o preservar dsa
Lazo de condensadores	Reemplaza una ecuación de voltaje por la ley del lazo derivada.	Relaciones del elemento necesarias para obtener DV, incluidos términos de capacidades variables.
Corte de bobinas	Reemplaza una ley de corrientes por la ley del corte derivada.	Relaciones para obtener DI y orientación de las contribuciones.
Cambio de conexión	Resuelve redistribución de carga o flujo antes del paso.	Conservación y compatibilidad del estado tras el cambio; no solo ecuaciones de evolución suave.
Fuente en lazo o corte	Usa su derivada declarada o calculada; cero si es constante.	Preservar esa contribución y su signo.
Punto de operación DC	Usa las leyes originales; las derivadas solas pueden quedar en `0 = 0`.	Elegir el conjunto de ecuaciones apropiado sin perder restricciones.

Actualmente parte del conocimiento reside en el generador y seguirá correspondiéndole la construcción de las leyes globales. En la propuesta, LOOP/CUTSET señalan los elementos a vigilar. Faltan las reglas de obtención de derivadas y de vigilancia de elementos de varios terminales, además de implementar y verificar este diseño.

2.17. 17. Cobertura del corpus y validación

El registro previo informa que se inventariaron 589 fuentes `.d`/`.ds` bajo `siru/corpus`, excluyendo directorios llamados `nousamos`, `libviejas` y `viejos`. El recorrido incluyó copias en directorios de construcción; no son 589 problemas independientes. Según ese registro, el lector procesó 586; 86 de ellas no tenían ramas. Esto fue una lectura e inventario, no una prueba de equivalencia numérica en `.dsa`.

Tipos encontrados: BAT, CB, CC, EB, ESIN, EV, JB, KE, KJ, LB, LNL, MQER, MW, MZER, RB, RC y SB. Un mismo nombre histórico puede tener distintas definiciones según la biblioteca del caso; no basta traducir por nombre.

Familia	Tratamiento en el diseño
RB y RC	Ecuaciones resistivas, incluidas las variantes no lineales reales.
CB, CC y CINC de la biblioteca	Distinguir almacenamiento en carga y capacidad incremental; preservar el estado correcto.
LB y LNL	Relaciones de corriente, derivada y voltaje según el modelo concreto.
BAT	Batería con carga interna y tensión dependiente de ella.
EB, JB, ESIN, EV	Fuentes y relaciones funcionales; revisar signos y ecuaciones de derivadas de las versiones antiguas.
MW, acoplamientos y auxiliares mecánicos	Modelos de varios terminales o composición con sus ecuaciones, sin imponer la estructura histórica de ramas auxiliares.
KE y KJ	Convolución con historia y parte instantánea cuando corresponda.
SB	Sección especial de llaves, fuera de los modelos de elementos.
MZER	Auxiliar histórico sin ley constitutiva; su función debe quedar en las ecuaciones del elemento o sistema que lo usaba.

Modelos compuestos nombrados en esas fuentes:

```
101M, 1N276, 1N754, 2N2219, 2N2369, 2N2369A, 2N2646,
2N3608, 2N4405, 2N706A, 2N9999AA, 2ORDER, C358E, DIODE,
FD300, GAIN, GSAT, JCNTRAN, MARX1, MOTOR, PDIODE, POWERDIODE,
SAMP, SCR, SPICE_SCR, SS1, T527Z12, THYRISTOR, TRAFO, TRANSISTOR
```

La biblioteca permanente añade otras definiciones. Cada dispositivo debe conservar sus relaciones y coeficientes particulares; no queda sustituido automáticamente por el diodo o transformador genérico mostrado arriba. Los modelos históricos donde D era incremento o P era valor previo requieren traducción explícita.

Tres fuentes no pudieron leerse: `corpus/corpus/beispiele/s138.d` (sentencia interpretada como unidades), `corpus/corpus/inventados/tipo-sin-modelo.ds` (tipo CZZ sin modelo) y `corpus/ngspice/examples/Monte_Carlo/OpWien.ds` (no se encuentra IMPEDANCE_XOP). No se afirma cobertura completa ni ejecución `.dsa` mientras esto y la traducción estén sin resolver.

2.18. 18. Evaluación frente al esquema actual

Criterio	Evaluación
Lectura y autoría	Mejora propuesta: nombres significativos, residuos explícitos y estados declarados.
Organización	Mejora propuesta: distingue modelo de elemento, instancia, funciones locales y topología; terminales e1, e2 distinguidos de nodos internos.
Expresividad	Busca conservar las capacidades actuales, incluidos modelos compuestos, tablas, memoria y cambios de conexión. Los ejemplos no demuestran todavía equivalencia completa.
Robustez y rendimiento	No se ha demostrado una mejora: dependen del generador, los métodos numéricos y su implementación.
Madurez	El esquema actual tiene implementación y casos probados. La propuesta todavía necesita cerrar detalles, implementar el lector y comparar resultados.

El diseño propuesto es más claro en los aspectos anteriores, pero no puede calificarse aún como superior en funcionamiento o cobertura. La comparación deberá verificar ecuaciones, signos, estados, cambios de conexión y resultados contra las bibliotecas actuales. La lista anterior distingue decisiones de diseño abiertas de tareas de implementación y validación; no se considera todo resuelto.

La mejora de claridad reside también en separar modelo, datos y operación; distinguir parámetros, estados y semillas; obtener automáticamente las variables; y reconocer terminales externos y nombres de instancias. No requiere al autor declarar una variable por separado ni conocer todos los detalles internos del armado de ecuaciones.

Este documento se organiza por conceptos y mantiene la comparación histórica aparte. La claridad del lenguaje no prueba por sí sola que su implementación sea más robusta, rápida o completa: eso debe verificarse mediante traducción y simulación.

2.19. Formatos de entrada: dsa, y nada más

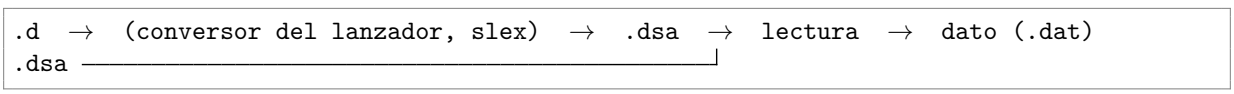
Tres reglas del autor, 10/11/2026, que reemplazan todo lo anterior sobre formatos de entrada.

Primera: el `.ds` no existe. No existe en decks, no existe en programas, no existe en ningún lado. Se olvida. Lo que quede de él es memoria, en `viejos`.

Segunda: la conversión de `.d` a `.dsa` está en el lanzador, y es el único que conoce el `.d`. No hay `.d` en ningún otro lugar, tampoco en SCEPTRE: los archivos históricos de SCEPTRE pasan al lanzador, y **SCEPTRE lee sólo `.dsa`**. Todo uso de un `.d` es a través de ese conversor, basado en slex. Un `.d` sigue siendo legible porque hay gente en el mundo que los tiene; lo que no se hace más es crearlos ni leerlos en otra parte.

Tercera: no se usa más python en el proyecto, salvo en SciPy. El traductor en python está en `slex/viejos/dsa-python` y nada lo invoca; el motor SciPy es la única excepción, por ser el integrador que se usa.

De ahí sale la forma de la cadena de lectura:



y no al revés. Hasta el 10/11/2026 era al revés: el `.dsa` se traducía a un `.ds` intermedio y lo leían las gramáticas escritas para la sintaxis del `.ds`. Eso es lo que hay que dar vuelta, y es el trabajo que queda:

	Trabajo
1	Que libdsa lleve el `.dsa` al dato **directamente**, sin intermedio. `dato.h` ya exporta la puerta que usan los demás (`dato_sentencia_origen`, con archivo y renglón), así que no hacen falta gramáticas nuevas: cambia quién las llama. Con eso el origen que anota el dato es el `.dsa` y su renglón.
2	El conversor `.d` → `.dsa` en el lanzador, basado en slex.
3	Que `d2datdinc` acepte `.dsa` y `.dat`, y nada más.
4	Los 81 `.ds` que quedan fuera de `viejos`: a `.dsa` los casos vivos, a `viejos` los que sólo existían para probar el lector del `.ds`. Y los `.d` al lanzador.
5	Las menciones de `.ds` en documentación y comentarios.

Lo ya hecho de esto: siru no corre python para leer y el traductor quedó en `viejos` (r7196); `CALCULA_DSA`, que cambiaba el lector, no existe más (r7199); y el netlist de SPICE sale del dato en lugar de un `.ds`, con lo que se fue el `--ds` de `d2datdinc` (r7202).

2.20. 19. Qué queda por concretar e implementar

Tipo	Trabajo
Gramática	Cerrar la escritura exacta de bloques y el alcance de funciones locales, sin reabrir los acuerdos de nombres y variables.
Generador	Obtención de derivadas y tratamiento de elementos de varios terminales en lazos y cortes. Por ahora son reglas internas del programa.
Traducción	Implementar ambos sentidos entre formatos actuales y dsa, preservando ecuaciones, datos, controles, signos y modelos.
Verificación	Probar ida y vuelta y simulación; el texto de regreso puede diferir aunque represente el mismo problema. No confundir igualdad textual con equivalencia del modelo.
Compatibilidad	Reutilizar Tabular, units y los mecanismos actuales de llaves y convolución. Implementar el punto con llaves según lo acordado.
Fuera del alcance actual	Organización y combinación entre distintos archivos de modelos, datos y operaciones.

2.21. 20. Registro de decisiones reemplazadas

Propuesta anterior	Acuerdo posterior
Declarar variables en VARIABLES	No hay declaración de variables; surgen de las ecuaciones, ramas y demás fuentes del modelo.
Bloque DERIVADAS con DX=derivada(X, tiempo)	Prefijo D y VARIABLE_INTEGRACION.
ARGUMENTOS obligatorios en cada elemento	Deducir los nombres de sus ecuaciones; no exigir esa lista redundante.
Declarar nodos y terminales en cada modelo	Deducir terminales por variables y ecuaciones; usar e1, e2, ... como externos en ramas internas y dar nodos y orden al conectar.
R1.tension, R1.corriente	VR1, IR1.
Ntiempo y otros prefijos N artificiales	tiempo y nombres en minúscula; no están reservados.
Crear PE o entrada para el voltaje de la batería	Usar VB1 directamente cuando corresponda.
Identificar parámetros por el prefijo P	PARAMETROS clasifica variables; P deja de ser un prefijo reservado.
Transformador construido obligatoriamente con vínculos entre devanados	Un único modelo de cuatro terminales con variables internas comunes.
Llave como rama común	Sección especial que cambia conexiones.
Sección UNIDADES	Unidades junto a números; unidades solicitadas junto a variables en RESULTADOS.
Resultados sin separación explícita entre entradas	Variables o pares variable–unidad separados por comas.

2.22. 21. Nota sobre organización de archivos

Una biblioteca es un archivo o conjunto de archivos que permite guardar y compartir contenido reutilizable. Funciones, modelos de elementos u otras partes pueden organizarse así; biblioteca no es una categoría matemática del modelo ni modifica el significado de su contenido.

2.23. 22. Referencias locales

- Guía para escribir un modelo: explicación previa y formatos actuales; las decisiones `.dsa` actualizadas se registran aquí.
- Arquitectura compartida (`../../../MODELOS-COMPARTIDOS.md`).
- Biblioteca principal de elementos (`../../d2datdinc/biblioteca/libraryori.siru`).
- Modelos permanentes de SCEPTRE (`../../d2datdinc/biblioteca/modellib.siru`).
- Generación de ecuaciones, lazos y cortes (`../../din/README.md`).

2.24. Selector de integración unificado

INTEGRA=[...] es el único selector público de métodos en .d, .ds y .dsa. En dsa se escribe directamente en la raíz del archivo, fuera de USOS/CONTROLES; cuando se separan M y DU, pertenece a M como método predeterminado. La opción `--integra` de Siru sustituye ese valor al preparar M para una corrida. El lector acepta los selectores antiguos y el verificador avisa con archivo y línea. Los generadores emiten MOTOR e INTEGRA en M, con la lista numérica de INTEGRA, sin serializar las marcas internas value/implicito. Los DU antiguos que

contengan INTEGRA se corrigen al utilizarlos: se traslada el método predeterminado a M y, si corresponde elegir otro método para una prueba, se usa `--integra`. No se agrega compatibilidad para conservar INTEGRA en DU. Cada número fija la receta completa, incluidas su tabla y su variante; las reservas futuras se diagnostican como no disponibles. El número se resuelve exclusivamente en el catálogo de MOTOR: el mismo número puede designar métodos distintos en motores distintos y se rechaza si el motor seleccionado no lo ofrece. Dentro de Siru los números son globales entre sus familias y se conservan los de GLM; 2/3 seleccionan ROS2/ROS3, 4/5 Euler IMEX/ARS222 y 6/7 Lie–Trotter/Strang. Las familias quedan como motores internos; los selectores anteriores se aceptan solo por compatibilidad de lectura y el traductor escribirá INTEGRA en M.

Una lista alterna métodos por pasos aceptados y vuelve al primero; un rechazo no avanza la lista. La lista puede alternar GLM/RK, BDF/Adams, ROS, IMEX y composición: los motores comparten estado e historia física aceptada. Se mantienen los requisitos del modelo y las capacidades de cada método; IMEX y composición aún requieren paso fijo. Seleccionar el método no pide por sí solo una evolución temporal. Las ecuaciones generadas por él siguen formando parte del sistema de cálculo preparado. INTEGRA=[-1] conserva la operación de un punto.

Las definiciones que cambian la preparación del integrador, como `IMEX_EXPLICITA`, `IMEX_IMPLICITA`, `PGLM_CAMPO_*`, `PGLM_PARTICION_*`, `SPLIT_*` y `SIRU_ETD_L`, pertenecen a `CONTROLES` de M y se rechazan en DU. M puede conservar las variantes `IMEX_EXPLICITA_IMEX_MRI` y `IMEX_EXPLICITA_MRI_IMPLICITO` además de `IMEX_EXPLICITA`: Siru elige la que corresponde al número de `INTEGRA` al preparar cada método, incluso cuando la lista alterna entre ambos. Paso y tolerancias son controles de ejecución y pueden quedar en DU. Las variantes se interpretan únicamente al seleccionar el método; no cambian el modelo fuente ni permiten que DU redefina sus ecuaciones.

Las llaves se verifican y estabilizan entre pasos, también con ROS, IMEX y composición. Dentro de las etapas y reintentos la conexión permanece fija. Al conmutar se resuelven los ajustes de lazos/cortes y se reinicia la historia; no se localiza la conmutación dentro del paso. No cambia la sintaxis de LLAVES ni la separación M/DU.

2.25. Selector del generador de ecuaciones

NETLIST=["..."] elige quién arma las ecuaciones a partir de la netlist. Se escribe como frase del deck en la raíz del archivo, fuera de USOS/CONTROLES, y cuando se separan M y DU pertenece a M. Los valores son `siru`, que es el de siempre (libdin, `slex/din/din.c`, por `d2datdinc`), y `mtk`, que usa ModelingToolkit.jl (`mtk/dsa2din.jl`, documentado en mtk/LEEME.md (`../../../mtk/LEEME.md`)). Sin la frase, el generador es `siru`.

Elegir el generador no cambia el modelo ni los usos: cambia de dónde salen las ecuaciones del mismo modelo. No es una opción de línea de órdenes (acuerdo del 09/11/2026): la elección pertenece al archivo, como las demás.

Las ecuaciones del método de integración las agrega siru en los dos casos, con el número que diga INTEGRA; el generador entrega el sistema del instante y, cuando hay degeneraciones, el del equilibrio. Un generador puede no cubrir todo el lenguaje: la falta debe informarse explícitamente, no sustituirse en silencio por otra formulación.

2.26. PARAPLOT en datos y usos

PARAPLOT pertenece a DU: no define ecuaciones, datos de cálculo ni métodos de integración. Puede repetirse con nombres diferentes; sin nombre recibe el número de su posición entre los PARAPLOT del documento. Los nombres repetidos se rechazan. En DSA cada bloque se cierra con FIN; en .d/.ds termina al comenzar otra sección o al finalizar el documento.

```
PARAPLOT evolucion
lineas G = TIME:X
```

```
titulo G = Evolución de X
FIN
PARAPLOT
puntos P = X:Y
FIN
```

El segundo bloque se llama 2. Siru y Sceptre transportan las órdenes junto con las tablas calculadas, sin ejecutarlas ni convertirlas en ecuaciones. Simplot las interpreta para dibujar. La traducción a M/DU conserva nombres, orden y valores de las órdenes. Los lectores C y Python usan `slex/dsa/dsa.slex`.

Prueba: `python3 -m unittest discover -s slex/viejos/dsa-python -p test_paraplot.py -v` desde la raíz de siru. Incluye varios bloques, nombres implícitos, separación histórica, ida y vuelta .d, lectura C y corridas de Siru con Calcula.

2.27. Cadena completa del proyecto

```
dia → lanzador → simusol → (siru | sceptre | spice | maxima) → simplot → gnuplot
```

Los cuatro motores son alternativas de cálculo: no se ejecutan uno después de otro. El lanzador coordina la cadena y selecciona el motor.

Etapa	Responsabilidad
dia	Crear el diagrama y guardar el archivo .dia.
lanzador	Coordinar la traducción, el motor elegido y la salida gráfica.
simusol	Interpretar el diagrama, sus conexiones y sus cuadros; preparar la descripción del sistema y de sus usos. Esta parte se trabaja en simusol.
siru, sceptre, spice o maxima	Calcular con el motor elegido. Los adaptadores llevan la descripción común al formato que necesita cada motor y recuperan sus resultados.
simplot	Recibir las tablas y PARAPLOT; preparar las instrucciones GNUPLOT. Si falta PARAPLOT, usar el dibujo implícito.
gnuplot	Ejecutar esas instrucciones para producir el dibujo.

La cadena es lineal: cada programa recibe el dato por stdin y entrega el dato para la siguiente etapa por stdout. Los diagnósticos, avisos y errores salen por stderr, separados del dato; no se mezclan con el documento ni con las tablas. Se selecciona un solo motor en esa posición de la cadena.

Puede comenzar leyendo un archivo ya guardado:

```
cat caso.dia | lanzador
```

Ese es el comienzo del flujo, que continúa por simusol, el motor elegido, simplot y gnuplot. El lanzador debe aceptar el .dia completo por stdin, sin exigir su nombre como argumento. Guardar un archivo intermedio es opcional; no es necesario para comunicar dos etapas.

PARAPLOT viaja desde la descripción de origen hasta Simplot, acompañado por los resultados. Los motores de cálculo no lo ejecutan ni lo usan en las ecuaciones. En la separación M/DU pertenece a DU. Cada etapa conserva la información destinada a las siguientes, aunque no la interprete.

Este es el esquema común de las cuatro rutas. Las pruebas actuales de transporte PARAPLOT verifican Siru y Sceptre; enumerar Spice y Maxima aquí no afirma que sus adaptadores ya implementen todo el recorrido.

2.28. Motor y salida de Maxima (2026-11-06)

`MOTOR = nombre` selecciona `siru`, `spice`, `maxima`, `sceptresimusolnuevo`, `sceptrenovender` o `sceptresimusolviejo`. En M/DU se escribe en el nivel superior de M, fuera de los bloques, junto a INTEGRA. En el documento histórico se escribe fuera de PARAPLOT. El lector conserva `GLOBAL.motor`. El lanzador usa Siru si falta la selección; `--motor` prevalece sobre el documento.

Dentro de PARAPLOT, `maxima = formulas` pide soluciones analíticas para Gnuplot y `maxima = tablas` pide muestreo numérico (predeterminado). Otros motores ignoran esta orden. Un bloque que solo contiene esa orden conserva los gráficos implícitos. En DU se cierra con FIN. No se admiten pedidos contradictorios de ambos modos en una misma corrida. Contrato, ejemplos y límites: ANALITICAS.md (`../../../simplot/doc/ANALITICAS.md`).

2.28.1. Informe HTML de PARAPLOT

`html = si` pide a Simplot guardar un informe HTML autocontenido con modelo, datos, resultados y gráficos. Los textos se presentan en cuadros copiables. `html = no` o su ausencia no lo pide. La orden no altera las ecuaciones ni el motor. En DU pertenece al bloque PARAPLOT, cerrado por FIN. Contrato: HTML.md (`../../../simplot/doc/HTML.md`).

2.29. 23. La lista completa de bloques y campos que el lector acepta

Medido el 17/11/2026 contra `slex/dsa/dsa.slex` (el único reconocedor de tokens del formato) y `slex/dsa/dsa.c` (el único lector: libdsa, que enlazan `d2datdinc`, las dos fases de SCEPTRE y siru, y cuyo flujo de tokens consume el generador mtk por `d2datdinc --tokens`). Las secciones anteriores documentan el modelo general; ésta completa lo que faltaba: la familia de interoperabilidad con SCEPTRE, la dinámica de siru y las declaraciones de preparación. Un bloque o renglón que el lector no sabe leer **corta nombrándolo, con su renglón**: nada se omite en silencio.

`dsa-verifica ARCHIVO.dsa` (o `d2datdinc ARCHIVO.dsa --verifica`) comprueba un archivo contra todo esto (sección 24).

2.29.1. 23.1 La familia SCEPTRE

Estos bloques conservan lo que un deck `.d` dice y el dato de siru no modela; viajan por el `.dsa` para que la vuelta `.d` → `.dsa` → `.d` sea completa. Los escribe `d2datdinc --dsa` y los consume la fase 1 de SCEPTRE.

- `DESCRIPCION:` (campo, ya documentado) — una cadena JSON; al venir de un `.d`, cada renglón de su `CIRCUIT DESCRIPTION` va **tal cual**, con su sangría y con el texto que siga a un `;`, porque SCEPTRE lista las tarjetas como están (17/11/2026).
- `CONTROLES_SCEPTRE` — bloque de cadenas JSON, una por renglón de `RUN CONTROLS` tal cual (`"X PLOT DIMENSION = 0"`). Va a `OTHERINFO.sceptre.run_controls` del dato.
- `ELEMENTOS_SCEPTRE` — bloque de cadenas, renglones de `ELEMENTS` que sólo SCEPTRE entiende; a `OTHERINFO.sceptre.elements`.
- `RUTINA_SCEPTRE:` (campo) — el código de la rutina de integración de SCEPTRE: 1 TRAP, 2 RUK, 3 XPO, 4 IMPLICIT. Equivale al renglón `INTEGRATION ROUTINE = nombre`; un `INTEGRA` explícito manda sobre él.

- **COTAS** — bloque de asignaciones `NOMBRE = JSON`: la cota de un parámetro o elemento, `{"min": a, "max": b}` o `{"porcentaje": p}` (manual de SCEPTRE 2.2.2/2.2.3). A `GLOBAL.cotas`.
- **AC** — bloque de asignaciones `NOMBRE = [re, im]`: el valor complejo de una fuente del análisis AC. A `GLOBAL.ac`.
- **SALIDAS_AC** — bloque de asignaciones `SALIDA = JSON` con la marca AC y la presentación de cada salida de `OUTPUTS`: `{"ac": "R"|"D"|"C"|"N", "rotulo": "U2", "plot": " GREEN"}` o `{"sinplot": 1}` (R RADIANS, D DEGREES, C COMPLEX, N NYQUIST; `plot` es el texto crudo después de `PLOT`). A `GLOBAL.salidas_ac` (16/11/2026).
- **SALIDAS_TARJETAS** — bloque de objetos JSON, uno por renglón: **cada tarjeta de `OUTPUTS` de un `.d`, tal cual**, con sus repeticiones, sus rótulos y sus salidas inexistentes a propósito: `{"vars": [["PX", "X"], ["PY"]], "ac": "C", "plot": " GREEN"}`. A `GLOBAL.salidas_tarjetas`; el escritor de `.d` las reescribe como estaban (17/11/2026).
- **RERUNS** — bloque de objetos JSON, una corrida repetida por renglón: `{"n": 2, "cambios": [{"linea": "...", "seccion": "FUNCTIONS"}, ...]}`. `n` es el de `RERUN DESCRIPTION (N)`; cada cambio lleva su subsección y su renglón tal cual —un *header card* (antes de toda sección) va con `"seccion": ""` y conserva su sangría—. A `GLOBAL.reruns`; la fase 1 los sirve a su lector de siempre (`RRPRO_LEE`). `n` cuenta corridas dentro de ese bloque; no es su número de orden. La selección del integrador se declara en M, también dentro de `USOS/RERUNS`, como `{"indice":2,"INTEGRA":[3,2,1]}`: `indice` identifica el segundo bloque de repeticiones de DU, contando desde uno, y cada número selecciona el método de una corrida de ese bloque en el catálogo de `MOTOR`. La lista debe tener exactamente `n` números enteros. No alterna métodos entre pasos de una misma corrida. DU conserva `n` y los cambios de datos; el lector reúne ambas partes en `GLOBAL.reruns`. Con `MOTOR=sceptre`, el ejemplo significa XPO, RUK y TRAP. La tarjeta antigua `INTEGRATION ROUTINE` en `cambios` se acepta con aviso; los escritores nuevos la normalizan a números en M.
- **SENSITIVITY, MONTE_CARLO, WORST_CASE** — bloques de conjuntos `(dependientes / independientes)`, uno por renglón, como los grupos del punto DC de SCEPTRE. A `GLOBAL.dc.{sensibilidad,montecarlo,peorcaso}`.
- **OPTIMIZATION** — acepta las dos formas: conjuntos `(dep / indep)` (a su propia lista, `analisis.optimization` del caso) y renglones de SCEPTRE como cadenas JSON (a `OTHERINFO.sceptre`).
- **POLOS_CEROS** (ya documentado en la sección 18) — conjuntos `(salida / fuente)`, su propia lista.
- **CONDICIONES_INICIALES** — bloque de asignaciones `NOMBRE = VALOR` o `NOMBRE = {"value": "20"}` (la forma del dato); equivalen al `INITIAL CONDITIONS` de siempre.

2.29.2. 23.2 La dinámica de siru

- **VALORES_RAMA** — corrige el valor de una rama del `CIRCUITO` (puede venir antes o después de él): `S1 = [["(T<1)+(T>3)", ["T"]], ...]`, una lista de valores, cada uno con su lista `V:` de variables; se unen con `&`. El valor de la rama recién queda al final del archivo.
- **ACOPLAMIENTOS** — las mutuas: `M12 = {"bobinas": ["L11", "L22"], "orden": n, "valor": ...}`. El `orden` es el del archivo original: los términos cruzados se suman en ese orden y la suma flotante lo nota, así que se conserva.
- **DERIVADAS** — `DCARGA = IE1`: la derivada del estado `CARGA` ya tiene nombre propio; debe ser `D` + un estado declarado.
- **ESTADOS_IMPLICITOS:** (campo) — estados cuya derivada ya es un parámetro del archivo (`DX` en `DEFINED PARAMETERS`): el estado no lleva su `D:`. Se declaran **antes** que `ESTADOS`.
- **EXTERNAS** — `FFFF = {"argumentos": 3, "fuente": "..."}`: una función externa (el `EXTERNAL` de `FUNCTIONS`).

- **RETARDOS** — `ALIAS = ESTADO, expresion`: el alias retardado de un estado, con su retardo τ como expresión. Exige la `HISTORIA` del estado.
- **HISTORIA** — `ESTADO = expresion`: la historia inicial para los retardos ($t \leq 0$).
- **DIFUSIONES** — `DIFUSION(estado, ruido) = expresion`: el término de difusión de una ecuación estocástica; el estado debe estar declarado y el par (estado, ruido) no se repite.
- **MECANICA** — la mecánica con restricciones: los campos `COORDENADAS:` y `MOMENTOS:` (listas de nombres) y las funciones `MASA_INVERSA(i,j) = expr`, `FUERZA(q) = expr`, `RESTRICCION(k) = expr`, `GRADIENTE(k,q) = expr`.
- **ECUACIONES_SIRU** — la ecuación anotada de siru, un objeto JSON por renglón con los campos `eq`, `vec`, `vars`, `indep`, `lineal`, `nucleo`, `res`, `tipo` (y ninguno más): la forma lineal lleva `vec` + `vars` (coeficientes y variables), la general lleva `eq`; `tipo` "N" es la omisión y no se escribe.
- **METODOS_INTEGRACION** — asignaciones `NOMBRE = JSON` con los datos de los métodos (van a la sección DATA del dato, por caminos). El generador de ecuaciones los ignora: no cambian el sistema.
- **LLAVES_RESISTIVAS** — asignaciones `NOMBRE = JSON`, bajo la clave `llaves_resistivas` del dato.
- **PARAMETROS_FIJOS** (dentro de `MODELO`) — asignaciones `NOMBRE = VALOR`: las constantes del modelo, las que en un `.ds` van en `CONSTANTS` y en el dato son `constants`. No son datos de un caso –eso es `DATOS`– sino constantes del modelo, y el orden en que se escriben es el de `lconstants`. Una constante declarada sin valor se avisa y no se escribe.
- **SEMILLAS** (ya documentado) — valores iniciales de variables: sólo ayudan a converger, no cambian el sistema.
- **UNIDADES** — bloque de renglones **tal cual** (cada uno entre comillas, como una cadena JSON): los de la sección `UNIDADES` de un `.ds`, `"TEMP: tempC, temperatura"` o `"plantilla=Generico"`. **La unidad que se declara es la del cálculo**, no la que decía el archivo original: el valor que el `.dsa` escribe ya está convertido, y declarar la de entrada lo convertiría otra vez. Lo que no se guarda, entonces, es en qué unidad estaba escrito el archivo.
- **RESULTADOS_GRAFICO** — bloque de renglones tal cual: la sección `RESULTADOS` de un `.ds`, la lista de resultados a graficar (`"TEMP(Interior,temperatura,tempF)"`). Se llama distinto porque el bloque `RESULTADOS` del `.dsa` ya son las salidas de siru (`OUTPUTS`).
- **GRAFICOS_ADICIONALES** — bloque de renglones tal cual: la sección `GRAFICOS ADICIONALES` de un `.ds`, órdenes que se agregan al último grupo de `PARAPLOT`.
- **FORMAS** — bloque de renglones tal cual: la forma, orientación, posición y tamaño de cada elemento para el dibujo, con los campos separados por **tabuladores** como en un `.ds`. La forma sola (`"RANT\tElectrico - vresistor"`) también vale, y el **bloque vacío** pide el dibujo automático. Cuando el bloque hace térmica una rama, el valor de esa rama se escribe **con su unidad de cálculo** (`= 278.15 K`) para que la conversión no lo cambie dos veces.
- **GNUPLOT** — bloque de renglones tal cual con las instrucciones de gnuplot; el encabezado de cada grupo, `"GNUPLOT <nombre>"`, va adentro del bloque, como en `PARAPLOT`.
- **DATABLOB** — bloque de renglones tal cual con el código Perl, una sentencia por renglón.

2.29.3. 23.3 Las declaraciones de preparación y los metadatos

No son ecuaciones ni datos: viajan en los metadatos del dato (`OTHERINFO.dsa_declaraciones`) y los usan los análisis de Calcula/Siru.

- **OBSERVABLES:** (campo) — los nombres observables del modelo.
- **VARIABLES:** (campo) — las variables de análisis.
- **LINEALIZADO** (marca sin valor, dentro de `MODELO`) — pide el análisis linealizado.

- `VARIABLE_INTEGRACION:` (campo) — la abscisa (por omisión `T`).
- `DOMINIO:` (campo) — el dominio por omisión del archivo, el `DEFAULTDOMAIN` de un `.ds`. `ELECTRICO` es lo que se supone, así que sólo se escribe cuando es otro.
- `SIRU:` (campo, dentro de `MODELO_ELEMENTO`) — los metadatos JSON del modelo propio (`tipo`, `valores`, `clase`, `if`, `ic`, `op`, `inv`): de ahí salen el tipo del `.ds` y los estados. La línea `ESTADOS:` del mismo bloque es **declarativa** (el estado ya viaja en el `ic/op`): se escribe para que un lector que no interprete ese JSON —el generador mtk— sepa cuál es el estado. El escritor (`d2datdinc --dsa`) nombra `Siru_<TIPO>` al modelo que define el propio deck, y así lo reconoce el lector: pasa cuando el deck declara un tipo nuevo y también cuando REDEFINE uno de la biblioteca (un `.ds` con su propio `EB`), caso en el que vale la definición del deck y no la de la biblioteca.
- `FORMATO:` (campo, dentro de `TABLA`) — el formato de las filas (`simul`, `pares`, u otro declarado).
- `INICIO_SALIDA:, INICIALIZACION:` (campos) — declaraciones de preparación de la corrida.

2.29.4. 23.4 Reservados

`OPERACION`, `FORMULAS`, `OPCIONES_SPICE` y `COMANDOS_SPICE` están en el léxico como nombres de bloque, pero el lector todavía rechaza su contenido: reservados para lo que documente una sección futura.

2.30. 24. El verificador

`dsa-verifica ARCHIVO.dsa...` pasa cada archivo por el lexer y el lector completos (libdsa, con sus validaciones) y dice `verifica` o el primer rechazo con su renglón; con `ARCHIVO.M.dsa` verifica también el `ARCHIVO.DU.dsa` que esté al lado. Sale 0 si todos verifican. Es parte del lector, en `slex/dsa` (`verifica.c`, la función `dsa_verifica` de libdsa), y se instala con d2datdinc; `d2datdinc ARCHIVO.dsa --verifica` hace lo mismo (17/11/2026). Como el lector es único, verificar con él **es** verificar contra la gramática (`dsa.slex`) y contra esta especificación.

3. Modelos de elementos en dsa

Este documento se incorporó al documento unificado Lenguaje dsa: modelo general y modelos de elementos. Esa es la fuente vigente para ambos temas; no se mantiene una segunda copia independiente.

El PDF actualizado es `AcuerdosGenerales_dsa.pdf`:

```
evince /home/dsa/svn/siru/slex/dsa/doc/AcuerdosGenerales_dsa.pdf
```

4. Gramática .ds y .d

Documento central del conjunto: Modelos compartidos: Calcula, Siru, SCEPTRE y Simplot (`../../../MODELOS-COMPARTIDOS.md`).

El archivo `.ds` es el eje que conecta los programas del proyecto Simusol: describe el problema físico o matemático, sus datos, ecuaciones, modelos, condiciones iniciales y controles, y puede incluir tanto el esquema como los gráficos de resultados. Puede escribirse directamente o prepararse a partir de un dibujo `.dia`. No exige que el problema sea un circuito.

Con `INTEGRA=[-1]`, el `.ds` contiene la descripción necesaria para construir el sistema de Calcula: Siru lo prepara, llama a Calcula para resolver un punto y escribe sus resultados en `.simul`. No se necesita una integración, un tiempo ficticio ni un `.din` escrito

por el usuario. Los archivos externos que se nombren —bibliotecas, tablas o funciones Fortran— deben acompañar al problema.

Esta referencia reúne sintaxis, significado, ejemplos y límites del lector `d2datdinc` y de los programas que consumen sus datos. Revisión: 29 de octubre de 2026. Describe las capacidades del árbol de fuentes revisado; una instalación anterior puede requerir actualización. Distingue las construcciones operativas de los encabezados heredados que sólo reconoce el lector.

Versión web en Simusol (`https://www.simusol.org/?Gram%C3%A1tica-.ds`) · Manual de Siru (`https://www.simusol.org/downloads/siru/docs/MANUAL.html`) · Slex (`https://www.simusol.org/?Slex`) · ExprTk (`https://www.simusol.org/?EXPRTK`) · Dibuja (`https://www.simusol.org/?Dibuja`) · Simplot (`https://www.simusol.org/?Simplot`)

4.1. 1. Un lenguaje compartido

Archivo o programa	Responsabilidad
`.dia` / Dia-Simusol	Dibujo de origen y datos del modelo; se transforma en una descripción textual que debe revisarse.
`.ds`	Descripción explícita del problema, controles y presentación; formato recomendado para modelos nuevos.
`.d`	Descripción procedente de SCEPTRE/Simusol, con las diferencias de sintaxis detalladas en §3. No toda extensión de Siru puede ejecutarse en SCEPTRE.
Slex / `d2datdinc`	Lee secciones y sentencias; produce el dato estructurado `.dat` y puede generar el sistema `.din`. La interpretación común de `UNIDADES` y `RESULTADOS` corresponde a esta etapa, antes de simular (§3.9 detalla la migración).
`.dat`	Dato JSON del problema: conserva modelos, ramas, ecuaciones y, si se declaran, `GLOBAL.formas` y `GLOBAL.simplot`. No es una tabla de resultados.
`.din`	Sistema preparado para Calcula y, para una corrida, sus descriptores de integración. No sustituye al `.ds` como documento que reúne también los dibujos y gráficos.
Calcula / ExprTk	Calcula resuelve las ecuaciones; ExprTk evalúa sus expresiones. Las funciones externas pueden implementarse en Fortran.
Siru	Coordina la construcción del sistema, la resolución de un punto o los pasos de integración, y escribe la tabla numérica.
`.simul`	Tabla de resultados: nombres, unidades cuando están definidas y números producidos por el cálculo.

Archivo o programa	Responsabilidad
Dibuja / `FORMAS`	Construye el esquema SVG a partir de las ramas y conexiones. Ayuda a comprobar visualmente la conversión `.dia` → `.ds`; no verifica por sí solo las ecuaciones ni los valores.
Simplot / `PARAPLOT`	Dibuja las tablas preparadas. PARAPLOT es opcional: sin él usa las columnas, nombres, magnitudes y unidades disponibles, sin reinterpretar las reglas físicas.

4.2. 2. Elegir el problema y ejecutarlo

Necesidad	Qué declarar
Resolver ecuaciones algebraicas sin ramas	Datos en `DEFINED_PARAMETERS`, incógnitas y semillas en `VARIABLES`, residuos en `EQUATIONS`, `INTEGRA=[-1]` en `RUN_CONTROLS`.
Integrar ecuaciones diferenciales sin ramas	Estados e iniciales, incógnitas derivadas, ecuaciones, familia de integrador, paso y condición de continuación; §10–11.
Simular un circuito eléctrico o térmico	Modelos en `ELEMENT_DESCRIPTION`, ramas en `ELEMENTS`, condiciones iniciales, integrador y salidas. La analogía térmica cambia el significado físico de las variables y sus unidades.
Resolver sólo el estado inicial de un circuito	Las mismas ramas y estados, con `INTEGRA=[-1]`; los estados quedan fijados en sus valores iniciales.
Calcular el punto de operación DC	Controles `RUN INITIAL CONDITIONS` de `.d`; no confundir equilibrio DC con fijar los estados iniciales mediante `INTEGRA=[-1]`.
Dibujar el esquema automáticamente	Añadir `FORMAS`, aunque quede vacía; el paquete Dibuja debe estar instalado.
Graficar datos ya calculados	Un `.ds` con secciones gráficas, o un `.paraplot`, puede entregarse a Simplot sin ejecutar Siru.

Instalación de los componentes mediante el instalador del proyecto:

```
curl -fsSLO https://www.simusol.org/downloads/installsumma.sh
sh installsumma.sh siru slex/d2datdinc dibuja/dibuja simplot
```

Ejecución habitual:

```
siru problema.ds
siru -nosimplot punto.ds
dibuja problema.ds
simplot problema.ds
```

Siru llama a Simplot al terminar, salvo `-nosimplot`. Dibuja y Simplot también funcionan por separado. En §17 hay ejemplos completos para descargar y comprobar. En los fragmentos, `...` significa contenido omitido y no debe copiarse como una sentencia.

4.3. 3. `.d` o `.ds`: cuál usar

Los dos describen lo mismo con las mismas secciones y las mismas líneas de elementos; el lector es uno solo (`d2datdinc`) y el sistema que sale es el mismo. Difieren en lo que sigue.

	`.d` (SCEPTRE)	`.ds` (Siru)
de dónde viene	lo que Simusol genera para SCEPTRE; el manual de SCEPTRE	el formato propio de Siru
tipo de rama	por la primera letra del nombre: `R1` es `RB`, `C3` es `CB`, `E1` es `EB`, `J1` es `JB`, `L2` es `LB`, `S1` es `SB`; un tipo propio que sea prefijo del nombre (`ESIN1`, `MW1`)	explícito, con guión: `RB-R1`, `CB-C3`, `SB-S1`
biblioteca de elementos	se incluye sola	hay que escribir `include .../libraryori.siru` (ruta relativa al archivo)
comentarios	`#`, y `;`	`#`; `;` separa sentencias
secciones	con espacio: `RUN CONTROLS`	con espacio o con `_`: `RUN_CONTROLS`
fin de la corrida	`STOP TIME = 5` o `T>5` en `RUN CONTROLS`	`TIEMPOFINAL=5` en `DEFINED_PARAMETERS` y `T<TIEMPOFINAL` en `CONTROL_EQUATIONS`
método de integración	`INTEGRATION ROUTINE = XPO` o sin rutina selecciona `INTEGRA=[216]`; también se admiten `TRAP`, `IMPLICIT`, `RUK` y `INTEGRA=[...]`.	INTEGRA=[112] en RUN_CONTROLS; números del catálogo global de §11.
paso	`STARTING STEP SIZE = 0.05`	`STARTING STEP SIZE = 0.05` (desde r1890; también `MINIMUM` y `MAXIMUM`), o `SIRU_PASO=[0.05]`
semilla de calcula	`SEMILLA VL2 = 0.1`	`SEMILLA VL2 = 0.1`, o `VL2=0.1` en `VARIABLES` si es de una rama
condiciones iniciales	`INITIAL CONDITIONS` (o `DEFINED PARAMETERS`)	`DEFINED_PARAMETERS`
modelos de SCEPTRE (`MODEL 101M (B-C-E)`, transistores de `modellib.siru`)	sí	no
tablas adentro del archivo (`TABLE 1`)	sí	sí (desde r1950), o por archivo aparte, `Table('tabla.txt',...)`

	`.d` (SCEPTRE)	`.ds` (Siru)
incluir otro archivo	SCEPTRE no tiene `include`, pero siru lo acepta: `include parte.d` mete otro `.d`, que se lee como `.d` (desde r2066); una biblioteca (`.siru`, `.ds`) se lee como `.ds`. Un `.d` con `include` ya no corre en SCEPTRE	`include otro.ds` (o `input otro.ds`): el otro archivo entra en ese lugar, con su ruta relativa al que lo incluye o al directorio actual; sirve para la biblioteca y para partes de un circuito
cadenas entre comillas	no	sí (`Table('...')`)

Cuál usar: `.ds` para lo nuevo —dice el tipo de cada rama, no lo adivina por la letra, y es el formato de Siru—; `.d` para correr lo que viene de SCEPTRE o de Simusol, o lo que usa sus modelos de transistor y sus tablas incorporadas. Lo que sigue muestra las dos formas donde difieren.

4.3.1. 3.1 Estructura y reglas de escritura

La siguiente referencia describe el formato `.ds` que lee `d2datdinc`, el lector basado en slex. Las secciones explican qué significa cada sentencia; una misma expresión escrita bajo otro encabezado puede significar otra cosa. El recorrido es:

```
problema.ds → d2datdinc → dato del problema (.dat)
                              ├─ sistema (.din) → Calcula → resultados (.simul)
                              └─ FORMAS + ramas → dibuja → esquema (.svg)
```

Siru coordina ese recorrido con `siru problema.ds`. Los archivos intermedios pueden ser temporales: no hace falta prepararlos a mano. `FORMAS` aporta información de dibujo; las ecuaciones se construyen con la descripción del sistema, sin depender de que exista el dibujo.

Regla	Escritura y significado
Encabezado	En una línea propia: `DEFINED_PARAMETERS` o `DEFINED PARAMETERS`. En encabezados se admiten mayúsculas/minúsculas, espacios, tabuladores y `_` entre palabras. Puede seguir un comentario.
Cuerpo	Desde el encabezado hasta el siguiente encabezado o el fin del archivo. Se pueden omitir las secciones que no se necesitan y volver a abrir una sección; esto no permite redefinir arbitrariamente los mismos nombres. Ninguna sección requiere una orden de cierre.
Fin	Basta el fin del archivo; no se añade `END` a PARAPLOT ni a las demás secciones. Por compatibilidad, el lector admite el `END` histórico, que termina la fuente actual y descarta lo que siga; no es un cierre de sección.
Sentencias	Una por línea, o varias separadas por `;`. En `.ds`, `;` **no** introduce un comentario.

Regla	Escritura y significado
Continuación	Una barra inversa \ al final une la línea siguiente. Terminar en `+`, `,` o `=` sin \ no pide continuación en `.ds`.
Comentarios	`#` o `//` hasta fin de línea; también `/* ... */`, con bloques anidados. Un bloque sin cierre es un error de lectura. Dentro de cadenas entre comillas no comienzan comentarios.
Inclusión	`INPUT ruta` o `INCLUDE ruta`, sin distinguir mayúsculas/minúsculas; se aceptan también `#INPUT` y `#include`. La ruta va sin espacios ni comillas y se busca relativa al archivo que incluye o al directorio de trabajo. Se admiten inclusiones anidadas y, al terminar cada fuente, se vuelve a la anterior. Un ciclo en la cadena activa se rechaza; volver a incluir un archivo después de terminarlo sí se permite. La sección activa puede continuar a través de la inclusión: conviene escribir encabezados explícitos al volver.
Nombres	Para parámetros e incógnitas: letra seguida de letras, dígitos o `_`, por ejemplo `TEMP`, `Y1`, `P_T`. Para tipo, rama y nodo se aceptan letras, dígitos y `_`, incluso un nodo numérico. No usar blancos dentro del nombre.
Caracteres no ASCII en expresiones	Slex admite UTF-8 en el archivo. Los nombres de variables y funciones, fuera de cadenas entrecomilladas, deben ser ASCII. Si un byte UTF-8 no reconocido aparece en una expresión, `d2datdinc` informa archivo, línea de origen, valor hexadecimal y posición en bytes, y detiene la conversión sin producir un DAT o deck parcial. No lo elimina ni altera la ecuación. Las cadenas, descripciones y comentarios conservan su texto UTF-8.
Mayúsculas	La tolerancia de los encabezados no se extiende a todas las claves. Escribir `RK`, `SIRU_PASO`, `V:` y `D:` como se documentan y mantener la misma escritura de los nombres. Los nombres del sistema generado pueden aparecer en minúsculas.
Números	Decimal con punto, signo opcional y exponente `e`/`E`: `12`, `-0.5`, `.001`, `2e-6`. La coma separa campos; no es separador decimal.
Cantidades con unidad	En expresiones: número, blanco y unidad GNU units, por ejemplo `10 kohm`, `100 uF`, `25 tempC`. El lector convierte la cantidad a SI. `10k` no es esta sintaxis; `C` es coulomb, `tempC` es Celsius absoluto. `CONSTANTS` y `DATA` tienen sus propias reglas indicadas más abajo.

Regla	Escritura y significado
Expresiones	Operadores explícitos: `+ - * / ^`, paréntesis, funciones y condiciones de ExprTk. Ejemplos: `2*(X+1)`, `sin(T)`, `if(T<1,0,1)`. No hay multiplicación implícita. Las comillas simples o dobles delimitan cadenas, como las rutas de `Table(...)`.

4.3.2. 3.2 Secciones del archivo

Sección	Contenido	Desarrollo
`CIRCUIT_DESCRIPTION`	Texto que describe el problema.	§4
`ELEMENT_DESCRIPTION`, alias `ELEMENT_DEFINITION`	Tipos de rama y sus ecuaciones; también pueden llegar mediante `include`.	§3.3 y §5
`ELEMENTS`	Ramas, nodos, valores y acoplamientos. Se omite en un sistema de ecuaciones sin circuito.	§3.3, §5–8
`CONSTANTS`	Constantes numéricas con nombre.	§3.4
`DEFINED_PARAMETERS`	Datos conocidos, expresiones auxiliares y valores iniciales de estados.	§3.4 y §10
`INITIAL_CONDITIONS`	Encabezado de compatibilidad con SCEPTRE. En un `.ds` nuevo escribir las condiciones en `DEFINED_PARAMETERS`.	§3 y §10
`VARIABLES`	Incógnitas y semillas para Calcula; también derivadas propias.	§3.4 y §10
`EQUATIONS`	Residuos que Calcula debe anular.	§3.4 y §10
`FUNCTIONS`	Funciones, tablas incorporadas o declaraciones `EXTERNAL`.	§3.5 y §9
`RUN_CONTROLS`	Selección del modo de resolución y opciones numéricas.	§3.6, §11 y §12
`CONTROL_EQUATIONS`	Condiciones para continuar una integración.	§3.6 y §11
`OUTPUTS`	Variables que se quieren observar y abscisa opcional.	§3.6 y §12
`FORMAS`	Símbolos de las ramas e información conservada del dibujo de Dia. Su presencia solicita el esquema.	§3.8

Sección	Contenido	Desarrollo
`DEFAULTDOMAIN dominio`	Dominio físico de las ramas sin forma; si se omite, `ELECTRICO`. No abre un bloque de registros.	§3.8.1
`DATA`	Datos estructurados de compatibilidad, como tablas de métodos.	§3.7
`DATABLOB`	Texto de compatibilidad con Siru antiguo; no es una sección general para programar el Siru actual.	§3.7
`RESULTADOS`, `UNIDADES`, `PARAPLOT`, `GRAFICOS`, `ACEPTAR_GRAFICOS`, `GRAFICOS_ADICIONALES`	Presentación de tablas y gráficos mediante Simplot.	§3.9
`SENSITIVITY`, `MONTE_CARLO`, `WORST_CASE` / `WORST-CASE`	Conjuntos de variables para los análisis DC correspondientes.	§11 ter–quinquies

El lector reconoce además encabezados históricos como `FORCED_VARIABLES`, `MODELS`, `MODEL_DESCRIPTION`, `MULTIELEMENT_DEFINITION`, `CONTROLS`, `ECUACIONES`, `OPTIMIZATION`, `CLEAR`, `RERUN_DESCRIPTION`, `CONTINUE` y `RE-OUTPUT`. Reconocer el encabezado no implica implementar su contenido en `.ds`: los modelos de SCEPTRE, reruns y continuaciones descritos en esta referencia usan su sintaxis `.d`. `FORCED_VARIABLES` no tiene traducción general en `.ds`; usar parámetros y ecuaciones explícitos. Las órdenes de la terminal antigua (`EXECUTE`, `PREPARE`, `SHOW`, `solve`, `plot`, etc.) no se ejecutan al leer un `.ds`.

4.3.3. 3.3 Gramática de ramas y modelos

En las reglas siguientes, las comillas indican texto literal, `{ ... }` significa repetición y `[ ... ]` significa parte opcional. `expresion` representa una expresión del lenguaje numérico, no un nombre literal. Los separadores se interpretan fuera de paréntesis y cadenas.

```
rama       = tipo "-" nombre "," nodo "-" nodo "=" valores [ "=" controles ]
valores    = pieza { "&" pieza }
controles  = pieza { "&" pieza }
pieza      = expresion [ "V:" nombre { "," nombre } ]
mutua      = nombre_M "," bobina "-" bobina "=" expresion
modelo     = tipo "(" cantidad ")" "=" campo { "," campo }
campo      = expresion | marca_modelo
```

Parte	Ejemplo y regla
Rama	`RB-R1, entrada-salida = 1000`. `RB` es el tipo; `R1`, el nombre usado en `VR1`, `IR1` y `FORMAS`.
Varios valores	En `.ds` se separan con `&`: `ESIN-E1, GND-1 = 5 & 2*pi*T`. Son `x1`, `x2`, etc., en el modelo. Las comas dentro de una llamada separan los argumentos de esa llamada.

Parte	Ejemplo y regla
Control de rama	El segundo `=` introduce el control histórico de la rama. Para una llave corriente usar el tipo `SB` y su valor de apertura/cierre (§6); otros controles pueden no tener traducción al motor actual.
Masa	Un circuito `.ds` usa `GND`. No sustituirla por `0`; esa compatibilidad pertenece a `.d`. Un sistema sin ramas no necesita masa.
Mutua	`M12, L1-L2 = 0.2`; las dos bobinas deben estar declaradas. El valor es la inductancia mutua, no el coeficiente de acoplamiento (§8).
Modelo algebraico	`RB(1) = VOLTAGE - x1*CURRENT`. Cada expresión del modelo es un residuo que se anula.
Estado de un modelo	`IC: CURRENT:CURRENT:DCURRENT` declara la asociación de estado/derivada del modelo de bobina; véanse los modelos completos de §5 y biblioteca.siru (`../../d2datdinc/doc/ejemplos/biblioteca.siru`).
Otras marcas de modelo	`IF: expresion`, `INV: expresion`, `OP: parte:parte`, `VARS: nombre:nombre`. Son campos del modelo; no encabezados del archivo. Las dos primeras intervienen en la definición y el signo de las fuentes. `OP` y `VARS` conservan información del modelo para su traducción.

La definición de los tipos debe estar disponible antes de construir el sistema. Se pueden copiar sólo los modelos utilizados dentro del `.ds`, como en los ejemplos de `dibuja`, o incluir la biblioteca. El valor de una rama no es su símbolo gráfico: la forma eléctrica, térmica o genérica se declara por separado en `FORMAS`.

4.3.4. 3.4 Parámetros, incógnitas, constantes y ecuaciones

```
parametro  = nombre "=" expresion [ "D:" derivada ]
variable   = nombre "=" semilla   [ "D:" derivada ]
constante  = nombre "=" numero [ unidad ]
ecuacion   = expresion { marca_ecuacion }
```

Sección y ejemplo	Significado
`DEFINED_PARAMETERS` / `PRES=101325`	Dato conocido: se entrega a Calcula con ese valor.
`VARIABLES` / `PV=1500`	Incógnita: 1500 es una semilla, no un valor impuesto. Calcula la determina mediante las ecuaciones.

Sección y ejemplo	Significado
`CONSTANTS / TKO=273.15`	Constante numérica, opcionalmente con unidad separada por un blanco: `LARGO=5 m`, `AMBIENTE=25 tempC`. Se convierte a SI y conserva metadatos de unidad; no admite expresiones generales como `K=2+3`.
`DEFINED_PARAMETERS / X=1 D:DX`	Valor inicial de `X` y nombre de su derivada. La derivada debe quedar determinada por el sistema. También se reconoce la relación convencional entre `X` y `DX` descrita en §20.
`DEFINED_PARAMETERS / POT=V*I`	Definición mediante expresión; no es un dato numérico independiente. Se traduce en una incógnita con su ecuación.
`EQUATIONS / PV-PW(TEMP)`	Calcula busca que el residuo sea cero: equivale a `PV=PW(TEMP)`. Escribir el residuo evita confundir una ecuación con la asignación de un parámetro.

Cada incógnita necesita la relación que la determina; las semillas ayudan a resolver el sistema pero no agregan ecuaciones. No repetir la misma declaración de parámetro, variable o constante: el lector avisa y descarta la repetida. En cambio, una sección reabierta puede agregar otros nombres.

Las marcas opcionales van **después** de la expresión y no se repiten en una sentencia:

Marca	Forma y uso
`V:`	`V:X,Y` declara los nombres asociados a la ecuación. Se permite `V: X, Y`; escribir la coma pegada al nombre anterior.
`LINEAL:`	Sin valor. La parte inicial pasa a ser una lista de coeficientes, alineada con `V:`.
`INDEP:`	Un valor: término independiente de la forma lineal. `1,-1 V:X,Y LINEAL: INDEP:2` representa `X-Y=2`.
`NONUCLEO:`	Sin valor. Marca una ecuación fuera del núcleo del sistema. Es información avanzada de compatibilidad; no usarla para una ecuación que Calcula deba resolver como parte del sistema.
`RES:`	Un nombre asociado al resultado de una expresión.
`TIPO:`	Un valor que clasifica la ecuación en el dato histórico; por ejemplo `TIPO:POST`. Su lectura no garantiza que cualquier clasificación tenga implementación en el motor actual.

4.3.5. 3.5 Funciones y tablas

```
funcion    = nombre_funcion "(" cantidad_o_argumentos ")" "=" expresion
externa    = "EXTERNAL" nombre_funcion "(" cantidad ")" ruta
tabla      = "TABLE" nombre_tabla
```

```
            numero "," numero { "," numero "," numero }
```

Forma	Ejemplo y alcance
Función por cantidad	`PW(1)=exp(x)`, `COC(2)=x/y`. Los argumentos formales son `x`, `y`, `z`, etc.
Función por nombres	`COC(A,B)=A/B`. El lector admite la lista explícita de argumentos; los nombres de función llevan letras y dígitos, sin `_`.
Uso	`COC(PV,PRES)` dentro de una expresión. Los argumentos van separados por comas y entre paréntesis.
Subcircuito `.d`	subcircuito.d (`../../d2datdinc/doc/ejemplos/subcircuito.d`)
Tabla incorporada	En `FUNCTIONS`, `TABLE 1` y después `0,0, 1,10, 2,20`. También puede ir todo en una línea: `TABLE 1, 0,0, 1,10, 2,20`. Se usa como `TABLE 1(T)`; §9 detalla las formas equivalentes y los saltos.
Tabla de resultados incorporada (prevista)	Una sección superior `TABLE` sin nombre si es la única tabla, o `TABLE NOMBRE` si hay varias; su contenido puede emplear cualquier formato textual que Tabular2 admita, no sólo el del ejemplo. §9 distingue esta ampliación de la sintaxis ya operativa.
Tabla externa	`Table('tabla.txt','TIEMPO','VALOR',T,1)`. La tabla referenciada debe acompañar al `.ds`; §9 explica columnas e interpolación.
Función externa	`EXTERNAL FGEN (2) fgen.f90`. Acepta entre 1 y 12 argumentos y requiere el fuente y las herramientas de compilación indicadas en §9.

4.3.6. 3.6 Controles, condiciones y salidas

```
control    = clave "=" "[" valor { "," valor } "]"
condicion  = expresion_logica { marca_ecuacion }
salidas    = [ abscisa ":" ] nombre { "," nombre }
```

Escritura	Efecto
`RUN_CONTROLS` / `INTEGRA=[-1]`	Un único punto resuelto por Calcula. No integra. Véase §12.
`INTEGRA=[112]`	Integración con el método global 112 (tabla histórica 12). Los métodos y sus límites están en §11.
`INTEGRA=[118,118,111]`	Secuencia de métodos; no mezclar `-1` con métodos de integración.
`SIRU_PASO=[0.01]` y `SIRU_ADAPT=[0]`	Paso fijo de 0.01. Sin la segunda opción, declarar un paso puede activar adaptación; véase §11.

Escritura	Efecto
`STARTING STEP SIZE = 0.01`	Forma compatible admitida también en `.ds`; `MINIMUM STEP SIZE`, `MAXIMUM STEP SIZE` y `SEMILLA VL2 = 0.1` son otras excepciones a la lista entre corchetes.
INTEGRA=[...]	Selector único de métodos de todas las familias; consultar el catálogo global de §11. No se mezcla con los selectores antiguos.
`CONTROL_EQUATIONS` / `T<TIEMPOFINAL`	Condición para **seguir**, no para terminar. Se detiene al dejar de cumplirse. Con `INTEGRA=[-1]` no provoca una marcha temporal.
`OUTPUTS` / `T:VC1,IR1`	Abscisa `T` y variables solicitadas para una corrida. No son ecuaciones ni valores impuestos. Conviene poner la lista completa en una línea, o usar \ para continuarla.
`OUTPUTS` / `TEMP,TBH,PRES,PV`	Lista de observables sin inventar una variable de tiempo. En modo de un punto la tabla de Siru contiene todas las variables del sistema de Calcula, incluidos los datos (§12).

Los valores de los controles no admiten listas anidadas; cada clave valida además sus opciones. Las repeticiones de una clave pueden acumular sus valores: declarar una sola vez el selector de método. Usar exactamente `INTEGRA=[-1]`, con el nombre INTEGRA y el valor entre corchetes; INTEGRA=-1 no es la escritura de este modo.

4.3.7. 3.7 DATA y DATABLOB

`DATA` conserva datos estructurados; no ejecuta Perl. Su gramática es:

```
asignacion = camino "=" valor
camino     = clave { [ "->" ] ( clave | indice ) }
clave      = "{" cadena "}"
indice     = "[" entero_sin_signo "]"
valor      = cuenta | cadena | "undef" | "[" [ valor { "," valor } ] "]"
```

Elemento	Regla
Ejemplo	`{'rungekutta'}->[3]->{'a'}=[[0,0],[1/2,0]]`. Se conservan listas, objetos, números y textos.
Cuentas	`+ - * / **`, paréntesis, `sqrt`, `exp`, `log`, `sin`, `cos`, `abs`, `int`, `atan2`. Aquí la potencia se escribe `**`.
Índices	`[0]`, `[3]`: enteros no negativos, sin signo. No confundir un índice con un valor: `[-1]` sí puede ser el **valor** de una lista.
Compatibilidad	`{'lnumrk'}=[-1]` aparece en archivos antiguos. Para pedir un punto al Siru actual declarar explícitamente `INTEGRA=[-1]` en `RUN_CONTROLS`; no depender de ese dato histórico.

Elemento	Regla
Errores	No se acepta código, un índice negativo, una cadena sin cerrar ni cambiar un objeto ya creado por una lista al recorrer un camino.

`DATABLOB` transporta el texto histórico y sólo tiene traducciones específicas, entre ellas el selector antiguo de método `numrk`. No constituye un lenguaje general admitido por el motor actual. Para un archivo nuevo usar ecuaciones, funciones y controles declarativos; el documento del lector (`../../d2datdinc/README.md#5-datablob-a-exprtk`) precisa el patrón heredado que reconoce.

4.3.8. 3.8 FORMAS: el dibujo dentro del .ds

El modelo explícito del octavo campo debe ser un identificador definido en `library.siru` o mediante `UNIDADES Modelo.Clase: unidad,magnitud`. Se valida al terminar, por lo que la definición local puede aparecer después. Un modelo desconocido o un identificador como `5` es un error, aunque no exista una rama asociada. Los símbolos históricos sin ese campo no requieren una definición de modelo.

`FORMAS` permite incluir en el propio `.ds` la información de `NOMBRE.formas`, el archivo que `dia2formas` extrae de un dibujo de Dia. No sustituye `ELEMENTS`: las ramas y sus conexiones siguen siendo la fuente del circuito. Puede aparecer entre las demás secciones; termina al comenzar otra sección o al llegar a `END`.

El formato completo tiene **siete campos separados por tabuladores reales** y un octavo campo opcional para el modelo de unidades, uno por renglón:

```
nombre<TAB>forma<TAB>orientacion<TAB>x<TAB>y<TAB>ancho<TAB>alto[<TAB>modelo_unidades]
```

`<TAB>` representa un tabulador, no los cinco caracteres de esa indicación ni la secuencia literal `\t`. Los espacios interiores pertenecen al campo. En este ejemplo las separaciones son tabuladores reales:

```
FORMAS
# nombre    forma   orientacion x   y   ancho   alto
R1  Electrico - hresistor   horizontal  4   1   3   1.5
masa    Electrico - tierra  fija    8   6   2   2
nodo 1  Electrico - nodo    fija    3   1   1   1
END
```

Campo	Contenido
`nombre`	Nombre de rama sin el prefijo de tipo (`R1`, no `RB-R1`), `masa` o el nombre de nodo de Dia (`nodo 1`).
`forma`	Nombre de la forma, por ejemplo `Electrico - hresistor`, `Termico - acumulador` o `Termico - 0gradosc`. Se conserva como texto.
`orientacion`	Texto de Dia, por ejemplo `horizontal`, `vertical`, `fija`, `hacia la izquierda` o con indicaciones de volteo. No es un código numérico.
`x`, `y`	Coordenadas de la esquina de la forma en Dia, en cm; `y` crece hacia abajo.

Campo	Contenido
`ancho`, `alto`	Dimensiones en las mismas unidades de Dia.

Los siete campos deben tener contenido y los cuatro últimos deben ser números finitos. Campos de más, campos vacíos o coordenadas no numéricas hacen fallar la lectura sin publicar un DAT parcial. Como formato abreviado también se admite `R1 Electrico - hresistor`, o dos campos separados por tabulador: sólo `nombre` y `forma`, sin geometría. No escribir siete campos separados por espacios: se interpretarían como la forma abreviada.

El lector conserva el orden, los nodos y las masas repetidas. En el `.dat` el bloque queda en `GLOBAL.formas`:

```
{"GLOBAL":{"formas":[{"nombre":"R1","forma":"Electrico - hresistor","orientacion":"
    horizontal","x":4,"y":1,"ancho":3,"alto":1.5}]}}
```

Es un fragmento del dato, no un circuito completo. `FORMAS` vacía produce `"formas":[]`; sin la sección, la clave no existe. Esa diferencia es deliberada:

Entrada	Acción de Siru
Sin `FORMAS`	Resuelve el problema sin solicitar esquema.
`FORMAS` vacía	Invoca `dibuja` para construir el esquema a partir de las ramas y conexiones, con símbolos automáticos.
`FORMAS` con registros	Invoca `dibuja` y le entrega las formas del dato ya leído.
`FORMAS` y `INTEGRA=[-1]`	Solicita el esquema y resuelve un punto con Calcula; dibujar no implica integrar.
`dibuja` ausente	Continúa el cálculo y deja constancia en `.log`.
Error de `dibuja`	Avisa y continúa el cálculo.

Para pedir el dibujo automático basta poner, antes de `END`:

```
FORMAS
END
```

`dibuja` elige símbolos con esta información y escribe `NOMBRE.svg` junto al circuito. Las formas internas prevalecen, por nombre, sobre las del archivo externo `NOMBRE.formas`; los nombres no cubiertos internamente pueden seguir tomando la forma externa. Las ramas restantes usan `DEFAULTDOMAIN` o, si se omite, los símbolos eléctricos automáticos; no se deduce su dominio por mayoría de formas. Para nombres repetidos se toma la primera forma; los registros de nodos se conservan pero todavía no seleccionan símbolos de nodo. `--electrico` permite pedir explícitamente la representación eléctrica al ejecutar `dibuja` por separado.

Alcance actual de la geometría: slex transporta orientación, posición y tamaño, pero `dibuja` usa ELK para distribuir el esquema. Todavía no reproduce las coordenadas, tamaños ni volteos originales de Dia. Siru activa este recorrido para `.d`, `.ds` y `.dat`, también al continuar una corrida; un `.din` no transporta `FORMAS`.

Ejemplo completo	Contenido
automatico.M.dsa (`../../../dibuja/ejemplos/automatico.M.dsa`, con su `.DU.dsa` al lado; era `automatico.ds`)	`FORMAS` vacía: esquema automático.

Ejemplo completo	Contenido
divisor.ds (`../../../dibuja/ejemplos/divisor.ds`)	Divisor de tensión con formas eléctricas y modelos incorporados.
termico.ds (`../../../dibuja/ejemplos/termico.ds`)	Circuito con formas térmicas.

Los esquemas de estos ejemplos se muestran también en la página de Dibuja. Para trasladar la información de Dia: ejecutar `dia2formas circuito.dia`, insertar el contenido de `circuito.formas` bajo `FORMAS` antes de `END` y conservar sus tabuladores. Para dibujar por separado: `dibuja circuito.ds` o `dibuja circuito.dat`; la segunda forma no necesita Calcula, Siru ni el lector `.ds`.

4.3.9. 3.8.1 DEFAULTDOMAIN: dominio de las ramas sin forma

Implementado en el lector, el DAT y la selección de símbolos de Dibuja. Es una cláusula del documento `.ds`, escrita en una línea propia, sin `=` ni cierre:

```
DEFAULTDOMAIN TERMICO
```

```
defaultdomain ::= "DEFAULTDOMAIN" dominio
dominio       ::= "TERMICO" | "MECANICO1" | "MECANICO2" | "ELECTRICO" | "HIDRAULICO"
```

Los valores se escriben en mayúsculas. La biblioteca distribuida define los cinco dominios. `MECANICO1` usa fuerza como análogo de tensión y velocidad como análogo de corriente; `MECANICO2` usa la correspondencia dual. No son movimiento traslacional frente a rotacional. `HIDRAULICO` usa presión y caudal volumétrico, no caudal másico. Las definiciones activas y sus alias están en `biblioteca/library.siru`; un documento puede ampliarlas con `UNIDADES Modelo.Clase: unidad,magnitud[,alias]`.

Dominio	V / fuente E	I / fuente J	R	C	L
`ELECTRICO`	Tensión, `V`	Corriente, `A`	`ohm`	`F`	`H`
`TERMICO`	Diferencia V: `K`; fuente E absoluta: `tempC`	Flujo de calor, `W`	`K/W`	`J/K`	Sin definición térmica
`MECANICO1`	Fuerza, `N`	Velocidad, `m/s`	Amortiguamiento b, `N*s/m`	Compliancia 1/k, `m/N`	Masa, `kg`
`MECANICO2`	Velocidad, `m/s`	Fuerza, `N`	Movilidad viscosa 1/b, `m/(N*s)`	Masa, `kg`	Compliancia 1/k, `m/N`
`HIDRAULICO`	Presión, `Pa`	Caudal volumétrico, `m^3/s`	`Pa*s/m^3`	`m^3/Pa`	`Pa*s^2/m^3`

Estas unidades se deducen de las ecuaciones genéricas `V=R*I`, `I=C*dV/dt` y `V=L*dI/dt`; los valores C y L no se convierten en una rigidez por cambiar de dominio. La numeración mecánica aquí adoptada toma como primera la analogía fuerza–tensión de los apuntes del proyecto (`termo/presesimusol/loops.tex`, cuadro «analogía»), y como segunda su dual. Como referencia externa sobre variables físicas y conexiones: Through and Across Variables, MathWorks (`https://www.mathworks.com/help/simscape/ug/through-and-across-variables.html`). Esa referencia no define la numeración propia de Simusol.

En térmica hay que distinguir la fuente de temperatura absoluta E (tempC por defecto) de una diferencia de potencial térmico V entre extremos, cuya unidad implícita es K. Una semilla `VR1=5` representa una diferencia de 5 K, no 278.15 K. La variable de una fuente E usa la definición de E; las declaraciones explícitas de UNIDADES siguen prevaleciendo. Para declarar otras temperaturas absolutas en Celsius, escribir su unidad expresamente.

Situación de una rama	Dominio que se aplica
Tiene una forma con modelo físico identificado	El modelo referenciado por esa forma; `DEFAULTDOMAIN` no lo reemplaza.
No tiene forma, aunque otras ramas sí la tengan	El indicado por `DEFAULTDOMAIN`.
No existe la sección `FORMAS`, o está vacía	`DEFAULTDOMAIN` se aplica a todas las ramas sin forma.
No tiene forma y se omitió `DEFAULTDOMAIN`	`ELECTRICO`.

El valor es único para el documento y no depende de que la cláusula aparezca antes o después de las ramas. No se infiere el dominio por mayoría de formas: las ramas sin forma usan el dominio declarado o, en su ausencia, el eléctrico. Los dominios desconocidos y las declaraciones contradictorias deben producir un error, no una elección silenciosa.

El dominio permite consultar en `library.siru` las magnitudes y unidades implícitas del modelo correspondiente, durante el procesamiento común anterior a Siru. No reemplaza las unidades escritas junto a un número ni las declaraciones de `UNIDADES`; tampoco cambia las ecuaciones de un tipo de rama explícitamente definido. No crea por sí solo una sección `FORMAS` ni solicita un dibujo.

Por ejemplo, este fragmento declara un dominio térmico por defecto y una excepción eléctrica explícita para R1; las demás ramas sin forma son térmicas:

```
DEFAULTDOMAIN TERMICO
FORMAS
R1 Electrico - hresistor
```

Campo del DAT	Contenido
`GLOBAL.defaultdomain`	Dominio declarado, o `ELECTRICO` si no se escribió la cláusula.
`GLOBAL.ramas.R1.dominio`	Dominio resuelto para esa rama.
`GLOBAL.ramas.R1.modelo_unidades`	Referencia a la definición de la biblioteca; el octavo campo opcional de FORMAS tiene prioridad sobre el nombre de la forma.
`GLOBAL.ramas.R1.unidad_modelo`	Unidad, magnitud y alias implícitos, cuando están definidos para esa clase de rama. Son metadatos de origen, no una afirmación sobre la unidad actual de `valor`.
`GLOBAL.unidades_modelo`	Definiciones leídas desde `library.siru` y las ampliaciones locales del documento.
`GLOBAL.ramas.R1.unidad_calculo`	Unidad SI del valor de la rama, unidad de origen y procedencia de la elección.
`GLOBAL.unidades_entrada`	Declaraciones de unidades de variables del documento, independientes de los gráficos.

Campo del DAT	Contenido
`GLOBAL.unidades_calculo`	Unidades SI de parámetros, semillas, constantes y variables físicas conocidas; Siru las conserva en sus tablas.

Dibuja ya no usa el conjunto predominante. Para `TERMICO` usa símbolos térmicos; para `ELECTRICO`, eléctricos. Los dominios mecánicos e hidráulico sin forma explícita usan por ahora un símbolo genérico, sin rótulos de unidades eléctricas. Las opciones explícitas `--termico` y `--electrico` siguen disponibles para el dibujo; esta última ignora las formas, como antes. Un archivo externo `.formas` afecta al dibujo, no cambia retrospectivamente el dominio físico que Slex resolvió en el DAT; para eso las formas deben estar en el `.ds`.

Conversión implementada antes de Siru: Slex normaliza a SI las entradas numéricas de ramas de un solo valor, `DEFINED_PARAMETERS`, semillas de `VARIABLES` y `CONSTANTS`. La procedencia se conserva separada de la unidad de cálculo: `unidad_modelo` describe el modelo; `unidad_calculo` describe el número ya normalizado. El `.din` se genera a partir del DAT convertido; cargar ese DAT no repite la conversión. Dibuja rotula los valores normalizados con su unidad de cálculo.

Caso ejecutable: defaultdomain.ds (`../../d2datdinc/pruebas/defaultdomain.ds`). Las pruebas `pruebas/probar-defaultdomain.py` cubren los cinco dominios, la omisión, FORMAS parciales y vacías, orden de cláusulas, bibliotecas locales, unidades explícitas, fin de archivo, CRLF, TABLE y errores. Las pruebas de Dibuja comprueban la selección y la ausencia de inferencia por mayoría.

4.3.10. 3.9 Unidades y resultados comunes; PARAPLOT para el dibujo

Contrato acordado el 11/10/2026: `UNIDADES` y `RESULTADOS` se interpretan en Slex (`ds2dat/d2datdinc`), al convertir el `.ds` en `.dat`, **antes de simular con Siru**, no solamente antes de dibujar. El lector resuelve las unidades y convierte los datos de entrada a las unidades del cálculo; deja interpretados los pedidos de `RESULTADOS` para aplicarlos cuando existan los valores de salida. Si Simplot recibe directamente un `.ds`, debe invocar ese mismo procesamiento común, sin otra implementación de las reglas físicas ni necesidad de simular una tabla de mediciones.

Información	Responsabilidad
`library.siru`	Unidades, magnitudes y alias por modelo físico, trasladados desde los cuadros de unidades de Dia. El lector consulta esta biblioteca, no los XML de Dia.
`FORMAS`	Nombre de rama, símbolo, referencia al modelo de unidades, orientación y geometría. No duplica las unidades de la biblioteca ni confunde apariencia con modelo físico.
`DEFAULTDOMAIN`	Selecciona el modelo físico de las ramas sin forma; por omisión, `ELECTRICO`. Las magnitudes y unidades implícitas se toman de `library.siru` y las entradas numéricas se convierten a SI, según §3.8.1.
`UNIDADES`	Unidad de origen y magnitud de cada variable, aplicables a toda la cadena antes del cálculo. Sintaxis: `VARIABLE: unidad, magnitud`.

Información	Responsabilidad
`RESULTADOS`	Selección y nombres de variables, magnitudes y unidades de salida, también sin gráficos. Admite `VARIABLE(nuevo_nombre,magnitud,nueva_unidad)`, `VARIABLE(nuevo_nombre,magnitud)`, `VARIABLE(nuevo_nombre)` y el nombre sin atributos. No agrega ecuaciones físicas.
`PARAPLOT`	Cómo dibujar los datos preparados: fuentes, coordenadas, series, ejes y paneles. Simplot conserva los metadatos recibidos; no reinterpreta las reglas físicas.
`RESULTADO GRAFICO` (previsto, no implementado)	Única excepción que permitiría a Simplot intervenir en las unidades: se aplicaría después de Siru, sobre los resultados recibidos y sólo para el gráfico. No modificaría las unidades del modelo, los datos del cálculo ni la tabla original.

Pendiente no prioritario: incorporar `RESULTADO GRAFICO`. Por ahora se documenta la intención; no se añade al lector ni a Simplot y no debe presentarse como sintaxis ejecutable. No es un sinónimo de `RESULTADOS`: este último pertenece al procesamiento común anterior a la simulación. PARAPLOT sigue describiendo cómo dibujar; no autoriza por sí solo a reinterpretar las unidades físicas.

PARAPLOT no es obligatorio. Si falta, Simplot usa lo que recibe: las columnas y sus nombres, magnitudes y unidades, incluidos los resultados preparados por el procesamiento común. Puede generar gráficos automáticos a partir de una tabla sola; la primera columna se usa como abscisa por convención y no tiene que ser tiempo. Una sola fila representa un punto. Este dibujo automático no traslada a Simplot la interpretación de `UNIDADES` ni de `RESULTADOS`.

Prioridad para interpretar un número	Unidad que se usa
1	La escrita junto al número.
2	La declarada para la variable en `UNIDADES`.
3	La del modelo físico referenciado por `FORMAS`, definida en `library.siru`; para una rama sin forma se usa `DEFAULTDOMAIN` o, si se omite, `ELECTRICO` (§3.8.1).
4	SI si no hay otra declaración; sin información dimensional no se inventa una magnitud.

Los datos se convierten a las unidades del cálculo antes de entrar al resolvedor; la conversión pedida en `RESULTADOS` se aplica a la salida, sin reinterpretar la entrada. Las conversiones usan Unidades/GNU units y el procesamiento de tablas usa Tabular2. `tempC` es temperatura absoluta con desplazamiento; las diferencias y unidades compuestas térmicas usan `K`, `K/W` y `J/K`.

Conversión de entradas: uso y alcance `UNIDADES` puede aparecer antes o después de los valores; no necesita `RESULTADOS` ni solicita gráficos. En las entradas del cálculo sus nombres son exactos: `E1:tempF,temperatura` declara la unidad del valor de la rama E1; `VE1:tempF,temperatura` declara la de la variable VE1, no la resistencia ni otros parámetros de la rama. Los nombres V/I (y Q/F cuando corresponden) seguidos del nombre de rama

heredan el modelo físico. Las derivadas heredan la unidad por segundo, sin el desplazamiento de temperatura absoluta: si VR1 está en `tempF`, DVR1 tiene escala 5/9 K/s, no suma 255.37 ni 273.15.

Entrada	Interpretación antes de Siru
Rama térmica E1 con valor `25`, sin otra unidad	`298.15 K`.
E1 con valor `77` y `E1:tempF,temperatura`	`298.15 K`; UNIDADES prevalece sobre el modelo.
E1 con valor `300 K`, aunque UNIDADES diga tempF	`300 K`; la unidad del número prevalece.
Parámetro sin unidad declarada ni modelo identificable	Se conserva como SI; no se inventa una magnitud ni se lo etiqueta como adimensional.
Entrada numérica `20+5` con unidad implícita tempC	Se convierte el resultado: `(20+5)+273.15`, no cada sumando.
Expresión `TEMP`, con TEMP ya normalizada	Se usa TEMP en SI, sin volver a sumar el desplazamiento.

Las expresiones con variables, llamadas a funciones o cantidades explícitas operan en SI; no se infieren unidades para cada coeficiente ni se realiza análisis dimensional simbólico. Para una función que produzca grados Celsius como números, la conversión debe escribirse explícitamente en su definición: no basta declarar la unidad del destino. Las conversiones no afines, si se usan, requieren un valor literal; las expresiones numéricas admiten conversiones afines, incluida temperatura absoluta. Una unidad inválida o dos declaraciones contradictorias impiden publicar un DAT parcial.

Un modelo con varios argumentos no tiene una unidad común deducible para todos ellos. En esos casos se escriben cantidades con unidad en cada argumento, o parámetros previamente normalizados con sus propias declaraciones `UNIDADES`; no se extiende automáticamente la unidad de R, C, etc. a todos los argumentos. Las tablas mantienen su procesamiento de columnas en Tabular2; este paso no asigna la unidad de una rama a todas las celdas de una tabla ni convierte de nuevo sus datos.

Ejemplo completo ejecutable: unidades-entrada.ds (`../../d2datdinc/pruebas/unidades-entrada.ds`). Declara TEMP=25 con unidad tempC, resuelve X=TEMP y produce ambos valores como 298.15 K. `pruebas/probar-unidades-entrada.py` comprueba el DAT, el `.din` y una ejecución real con Siru/Calcula; Siru conserva K en la cabecera de la tabla para evitar una segunda conversión en los consumidores.

Salida común implementada: el lector publica `GLOBAL.resultados` aunque no haya PARAPLOT. Siru entrega la tabla con unidades de cálculo a `preparar-resultados` de Tabular2 antes de publicarla, incluso con `-nosimplot`. Allí se aplican selección, orden, nombres, magnitudes y unidades de salida. La marca por bloque evita aplicar otra vez el mismo contrato. El contrato de transporte y ejemplos de consola (`../../../tabular2/doc/RESULTADOS.md`) describe la correspondencia de columnas y los límites del exportador. El motor gráfico se unifica por separado; debe consumir estos resultados, no reinterpretarlos.

En el transporte histórico, estas secciones se guardan en `GLOBAL.simplot`; Simplot las procesa cuando los resultados están disponibles. Ese alojamiento de `UNIDADES` y `RESULTADOS` es transitorio.

Sección	Ejemplo
RESULTADOS	TEMP(Interior,temperatura,tempC),PV pide columnas con leyenda, magnitud y unidad de presentación opcionales.
UNIDADES	TEMP:K,temperatura declara el origen cuando la tabla no lo trae. No cambia por sí sola las ecuaciones ni convierte números del sistema.
PARAPLOT / GRAFICOS / ACEPTAR_GRAFICOS	archivo F = caso.simul, lineas G = F,1:2,3, titulo G = Resultado: una orden por sentencia.
GRAFICOS_ADICIONALES	Agrega gráficos a los pedidos principales.
Controles asociados	SIMPLOT_VERSION=[2] selecciona el contrato tipado; ANADIR_GRAFICOS=[1] suma los pedidos explícitos a los automáticos. Van en RUN_CONTROLS.

Las órdenes, su prioridad y las conversiones se detallan en los contratos DAT/PARAPLOT (../../../simplot/doc/DAT-PARAPLOT.md) y DAT/Simplot v2 (../../../simplot/doc/DAT-SIMPLOT-V2.md). Las unidades de presentación no deben confundirse con las cantidades con unidad de §3.1: estas últimas sí se convierten a SI al construir el problema.

4.3.11. 3.10 Cómo comprobar la lectura

Con el lector instalado, d2datdinc -v problema.ds -o problema.dat genera el dato y muestra advertencias de sintaxis o compatibilidad. Desde la raíz del árbol de fuentes:

```
slex/d2datdinc/d2datdinc -v \
  -c slex/d2datdinc/d2datdinc.slex \
  -b slex/d2datdinc/biblioteca problema.ds -o problema.dat
perl siru/bin/siru problema.ds
```

Algunas construcciones históricas mal escritas se descartan con aviso; por eso generar un .dat no prueba por sí solo que todo lo escrito haya sido utilizado. Revisar los avisos de -v. Siru rechaza un sistema con traducciones pendientes. Para comprobar una regla en el código, la gramática léxica está en d2datdinc.slex (../../d2datdinc/d2datdinc.slex); las gramáticas de sentencias, en parametros.y, ecuacion.y, declaraciones.y, control.y, data.y, edef.y y piezas.y del mismo directorio. dato.c implementa la interpretación de secciones y de FORMAS.

4.4. 4. Un archivo mínimo

Un RC alimentado por una batería, con la resistencia dependiendo del voltaje del capacitor. Da lo mismo en los dos formatos.

.d:

```
CIRCUIT DESCRIPTION
  Un RC con resistencia no lineal
ELEMENTS
 E1, GND-1 = 1
 R1, 1-2 = 1+0.1*VC1
 C1, 2-GND = 1
OUTPUTS
VC1, IR1
INITIAL CONDITIONS
```

```
QC1 = 0
RUN CONTROLS
INTEGRATION ROUTINE = TRAP
STOP TIME = 2
END
```

`.ds`:

```
include biblioteca.siru
CIRCUIT_DESCRIPTION
  Un RC con resistencia no lineal
ELEMENTS
 EB-E1, GND-1 = 1
 RB-R1, 1-2 = 1+0.1*VC1
 CB-C1, 2-GND = 1
OUTPUTS
T:VC1,IR1
DEFINED_PARAMETERS
T=0
TIEMPOFINAL=2
QC1=0
RUN_CONTROLS
INTEGRA=[112]
CONTROL_EQUATIONS
T<TIEMPOFINAL
END
```

En `.d`, la línea de un elemento es `NOMBRE, NODO1-NODO2 = VALOR`; en `.ds`, `TIPO-NOMBRE, NODO1-NODO2 = VALORES`. La masa es `GND`. Un `.ds` tiene que tenerla: sin `GND`, o con un nodo `0`, `d2datdinc` se para y pide ponerla. Un `.d` es la entrada de SCEPTRE, que no pide masa, y se acepta sin `GND`: la masa es el nodo `0`, o si no lo hay el nodo más chico (los números por su valor, antes que los nombres), y se avisa por la salida de errores (`el .d no tiene nodo GND: la masa es el nodo 1`). La corriente de la rama va del primer nodo al segundo y su tensión es `u(NODO1) - u(NODO2)`. Una fuente de tensión `E1, GND-1 = 1` pone el nodo 1 a +1 V.

4.5. 5. Los elementos

4.5.1. 5.1 La biblioteca (`slex/d2datdinc/biblioteca/libraryori.siru`)

tipo	qué es	modelo	estado
`RB`	resistencia	`V = x1·I`	
`CB`	capacitor, `x1` la capacidad **total** `Q/V`	`dQ/dt = I`, `Q = x1·V`	la carga `Q`
`CINC`	capacitor, `x1` la capacidad **incremental** `dQ/dV`; `CINC1, 1-GND = ...`	`I = x1·dV/dt`	el voltaje `V`
`LB`	bobina	`V = x1·dI/dt`	la corriente `I`
`EB`	fuente de tensión	`V = x1` (`E1, GND-1 = 1` pone 1 V en el nodo 1)	
`JB`	fuente de corriente	`I = x1`	

tipo	qué es	modelo	estado
SB	llave	V·x1 + I·(1-x1) = 0; x1 es 1 cerrada, 0 abierta	
ESIN	fuente senoidal, ESIN1, GND-1 = A, T	V = A·sin(x2), con su derivada	
MW	devanado de un transformador con núcleo, MW1, 2-GND = N, DPHI1	V = N·dPHI/dt	
LNL	bobina no lineal, LNL1, 2-3 = L, n	L·dI/dt = V^n	I
RC, EC, JC	lo mismo que RB, EB, JB		

Los dos capacitores. Con capacidad constante CB y CINC dan lo mismo. Con una capacidad que depende del voltaje (o de una corriente que depende del voltaje) no, porque el número significa otra cosa:

- en CB es la capacidad total, Q = C·V; la corriente es dQ/dt, que incluye el término V·dC/dt;
- en CINC es la capacidad incremental, I = C·dV/dt. Es el capacitor de SCEPTRE, y es lo que dan las fórmulas de un transistor: la capacidad de juntura y la de difusión (CE = 5 + 70·J1) son dQ/dV.

En un .d de SCEPTRE, una rama C... usa capacidad diferencial por defecto. El calificador TOTAL declara capacidad total. No hace falta renombrar las ramas del archivo original:

Declaración	Ley interpretada
C1, 1-GND = 1+3*VC1^2	Diferencial: I = (1+3*V^2)·dV/dt.
C1, 1-GND = TOTAL (1+VC1^2)	Total: Q = (1+V^2)·V; representa la misma ley del ejemplo anterior.
CB-C1, 1-GND = ...	Tipo explícito: conserva la ley total de CB.
CINC-C1, 1-GND = ...	Tipo explícito: conserva la ley diferencial de CINC.

Si el lector demuestra que la capacidad es constante, conserva internamente CB: ambas leyes coinciden y la carga sigue disponible como estado para los lazos y las llaves. Para una capacidad diferencial variable selecciona CINC. Los tipos explícitos y los modelos definidos por el usuario conservan su contrato. TOTAL debe ser un calificador separado; una llamada como TOTAL1(VC1) sigue siendo una llamada a función o tabla.

El tipo resuelto se conserva al traducir a .dsa. Al exportar un CB a .d, se escribe TOTAL para conservar su ley también en la vuelta. Pruebas numéricas y de ida y vuelta: python3 siru/tests/capacidad_sceptre.py; ejemplo de ambos tipos: slex/d2datdinc/pruebas/sceptre/dos-capac

Lazos con capacidad diferencial: la ley derivada del lazo usa directamente dV/dt de los elementos cuyo estado es el voltaje; los elementos con carga como estado conservan su tratamiento anterior. Pueden convivir ambos tipos. No se introduce una carga ficticia Q=C·V.

Caso comprobado	Resultado
Dos capacitores en paralelo, uno diferencial `1+3V^2` y otro constante `2`	La tensión y el reparto de corriente coinciden con la solución analítica independiente; prueba `test_lazo_diferencial_y_total`.
`sceptre2novender/corpus/a02.d`	Llega a `t=495` con las capacidades diferenciales originales y condiciones iniciales DC. Al refinar los pasos de ambos motores, la diferencia máxima baja de aproximadamente 1,5 % a 0,027 % de la escala; ver `corpus/errores-esperados/CAPACIDADES-Y-A02.md`. No se afirma igualdad exacta.
`sceptre2novender/corpus/d04.d`	Completa la corrida y sus dos RERUN con las capacidades diferenciales originales. Esto no certifica equivalencia de sus análisis estadísticos con SCEPTRE.
`corpus/corpus/llaves/lazo-capacitores-diferenciales`	Al encrucillada en `T=1`, conserva la carga integrada: para `Cdif=1+3V^2`, usa `Q=V+V^3`. La prueba verifica `V^3+3V=2` y tensiones finales iguales.
Capacidad tabulada con cambio de pendiente	El trasvase coincide con la integral independiente de los tramos; prueba `test_trasvase_tabla`.

Saltos instantáneos. Para la ley constitutiva de `CINC`, si la capacidad normalizada depende solo del voltaje propio y parámetros constantes, el armador obtiene el cambio de carga como Δ`Q=`$\int$`[Vantes,Vdespués] C(v)dv`. Esta diferencia entra en la conservación de carga de los nodos, junto con las cargas de los capacitores totales. El estado de `CINC` sigue siendo el voltaje. Se admiten funciones y tablas; las corrientes de diodos en paralelo se sustituyen por sus expresiones en el voltaje correspondiente cuando el armador puede hacerlo.

Control de la integral numérica	Valor
Método	Simpson adaptativo, con 16 intervalos iniciales y subdivisión local.
Tolerancia estimada por intervalo inicial	`1e-12/16 + 1e-10*abs(integral_inicial_del_intervalo)`.
Límites de trabajo	30 subdivisiones de profundidad y 65536 iteraciones.
Falta de convergencia o dominio inválido	No se entrega un trasvase como si fuera válido: falla la evaluación numérica.

Dependencias cruzadas. La trayectoria del salto puede quedar determinada por las ecuaciones del circuito. En un lazo simple cuyas ramas no pertenecen a otros lazos, las corrientes impulsivas tienen una única circulación: `Ii=si*`λ, `Ij=sj*`λ. Si `Ci` depende de `Vj` y la capacidad de referencia es `Cj(Vj)`, resulta `dVi=(si/sj)*(Cj(Vj)/Ci(Vj))*dVj`. El armador integra ese cociente y conserva la relación entre los estados anteriores y posteriores, junto con las restantes ecuaciones del circuito. Los signos se obtienen de la orientación de las ramas. La referencia puede ser un `CINC` local o un `CB` constante con ley reconocida `Q=C*V`.

Caso	Verificación
`corpus/corpus/llaves/lazo-capacitores-cruzados.ds`	`C1=d+3*V^2`, `C2=2`: al cerrar la llave, `V1=V2=V` y `V+2/sqrt(3)*atan(sqrt(3)*V)=1`. También se prueba invertir la orientación de una rama y reutilizar el modelo para calcular un solo punto con distintos datos.
Triángulo `CC1/CC2/CRB` de `d04.d`	Se generan las cuatro relaciones cruzadas de los transistores. Una perturbación de 0,1 V comprueba un trasvase no nulo y la conservación de carga.

Pendiente: lazos que comparten ramas y dependencias en varias variables de estado requieren un tratamiento más general. Cuando no se deduce la relación acoplada, permanece la comprobación conservadora de compatibilidad de la red capacitiva: los estados deben coincidir antes y después dentro de `1e-8*(1+abs(estado_previo))`. Es una limitación de implementación, no una demostración de que falten datos en el circuito. Los auxiliares de esa comprobación no se interpretan como carga física; tampoco se transforma una capacidad diferencial en total.

Una condición inicial en `DEFINED_PARAMETERS` no convierte una variable de estado, ni un voltaje o corriente de rama, en parámetro constante para este análisis.

Los nombres `R`, `E`, `J`, `S`, `L`, `C` sin la `B` son de una biblioteca vieja por incrementos y no sirven para `siru`: una `C` o una `L` así en un `.ds` da el mensaje «modelo por incrementos». Usar `CB`, `LB`.

4.5.2. 5.2 Lo que un elemento puede nombrar en su valor

en el archivo	qué es
un número	el valor, constante
`PL`	un parámetro de `DEFINED PARAMETERS` (`PL = 22`); dos ramas que nombren `PL` comparten el valor
`VC1`, `IR1`, `QC1`, `IL1`	la tensión, la corriente, la carga, la corriente de otra rama (o de la misma): así se escriben los elementos no lineales y las fuentes controladas
`T`	el tiempo
`DT`, `TICK`	el paso en uso y la cantidad de pasos dados (los pone el simulador, el archivo no los fija)
`QC(PC,VC1)`	una función de `FUNCTIONS`
`T1(T)`, `TABLE 1 (T)`, `Table('tabla.txt','X','Y',T,1)`	una tabla (sección 9)
`sin`, `cos`, `exp`, `log`, `abs`, `sqrt`, `if(c,a,b)`, `<`, `>`, `and`, `or`...	lo que entiende ExprTk

Las expresiones van en notación de ExprTk: la multiplicación es `*` y no se sobreentiende (`2*(x+1)`, no `2 (x+1)`); `^` es la potencia; una condición vale 1 o 0, así que `(T<1)+(T>3)` es «antes de 1 o después de 3».

Ejemplos en sintaxis `.d` (las explicaciones a la derecha son comentarios de documentación; en un archivo `.ds` requieren `#`, y las ramas, su tipo explícito):

```
R1, 1-2 = 1+0.1*VC1             resistencia que depende del voltaje del capacitor
C1, 2-GND = 1+0.5*VC1           capacidad que depende de su propio voltaje
```

```
E2, GND-3 = 2e5*VR1              fuente de tensión controlada (amplificador)
J2, GND-3 = 0.98*IR1             fuente de corriente controlada
E3, GND-4 = 0.5+0.2*sin(3*T)     fuente que varía en el tiempo
ESIN1, GND-1 = 1, 6.28*T         la senoidal de la biblioteca: amplitud y fase
J3, GND-5 = 1e-13*(exp(38*VJ3)-1)  un diodo: fuente de corriente función de su tensió
   n
```

4.5.3. 5.3 Tipos propios

Un modelo nuevo se declara en `ELEMENT DEFINITION` (`.d`) o `ELEMENT_DESCRIPTION` (`.ds`, que es la sección de una biblioteca): el nombre con la cantidad de parámetros, y las ecuaciones en `VOLTAGE`, `CURRENT`, sus derivadas `DVOLTAGE`, `DCURRENT`, y la carga `CHARGE`, `DCHARGE`; `x1`, `x2`... son los valores que da la línea del elemento, separados por `&` en `.ds` y por comas en `.d`.

```
ELEMENT DEFINITION
   EV(2) = IF: VOLTAGE, VOLTAGE + x1, DVOLTAGE + x2     fuente con su derivada
   MZER(1) = 0                                           elemento nulo
   RQ(1) = VOLTAGE - x1 * CURRENT^3                      resistencia cúbica
```

- La derivada que nombra el modelo dice cuál es el estado: `DCURRENT` hace de la corriente un estado (una bobina), `DCHARGE` de la carga (un capacitor), `DVOLTAGE` del voltaje. Con dos ecuaciones, la segunda *da* la derivada y no hay estado (es lo que hace `ESIN`: una fuente con su derivada conocida).
- `IF: VOLTAGE` identifica una fuente de tensión en el dato. El signo de la ecuación y la marca `INV:` forman parte de su definición; para conservar la convención de EB copiar su modelo completo, incluido `INV:VOLTAGE`.
- Un tipo cuyo modelo es `0` no aporta ecuación: su tensión y su corriente quedan libres para que las fijen ecuaciones propias (un transformador ideal, un girador: `slex/d2datdinc/pruebas/nulo.d`).
- En un `.d` el tipo se reconoce por ser prefijo del nombre (`EV1`, `MZER2`).

4.5.4. 5.4 Modelos de SCEPTRE (sólo `.d`)

`MODEL 101M (B-C-E)` bajo `MODEL DESCRIPTION`, o los permanentes de `slex/d2datdinc/biblioteca/model` (`T1, 2-3-1 = MODEL 2N2222 (PERM)`): subcircuitos con patas que se expanden donde se usan; cada elemento interno lleva el nombre de la instancia como sufijo (`CET1`). Diodos: `DIODE EQUATION (Is, T)` es `Is*(exp(T*VJ)-1)`. Ejemplos en `slex/d2datdinc/pruebas/sceptre/modelo.d` y `diodos.d`.

La definición de un subcircuito `.d` se abre bajo `MODEL DESCRIPTION`; `CIRCUIT DESCRIPTION` devuelve la lectura al circuito principal. No poner `END` entre ambos, porque termina la fuente. Ejemplo mínimo de modelo resistivo:

```
MODEL DESCRIPTION
MODEL DIVISOR (TEMP) (A-B-M)
ELEMENTS
R1,A-M=1000
R2,M-B=1000
CIRCUIT DESCRIPTION
Divisor encapsulado
ELEMENTS
E1,GND-1=10
T1,1-GND-2=MODEL DIVISOR
OUTPUTS
VR2T1
RUN CONTROLS
```

```
RUN INITIAL CONDITIONS ONLY
END
```

Parte	Regla de modelos `.d`
`MODEL NOMBRE (TEMP) (A-B-M)`	Modelo local con terminales externos en ese orden; se admite la cabecera sin `(TEMP)`.
`T1,1-GND-2=MODEL DIVISOR`	Conecta A a 1, B a GND y M a 2. Deben coincidir la cantidad y el orden de terminales.
`MODEL 2N2222 (PERM)`	Selecciona la biblioteca permanente `modellib.siru`, cargada por el lector de `.d`.
`(CHANGE R1 = 2000)`	Modifica elementos o parámetros de la instancia; las expresiones de cambio se conservan tal como se escriben.
Nombres interiores	R2 en T1 pasa a R2T1; un nodo interno 7 pasa a 7T1. También se distinguen parámetros, funciones y tablas; GND sigue siendo común.
Observación	Pedir `VR2T1` desde el circuito principal. Los `OUTPUTS` internos no se agregan automáticamente.
Ámbito	Se busca primero el modelo local, salvo `(PERM)`. Un modelo ausente no se sustituye por otro. Esta expansión de subcircuitos es una capacidad `.d`, diferente de los tipos de rama de `.ds`.

4.5.5. 5.5 Convolución: una caja descripta por su respuesta (K)

Una parte del circuito se puede dar por cómo responde en lugar de por sus elementos: su **respuesta al impulso** `h(t)`, por tabla o por ecuación. Es la convolución de SCEPTRE (manual, volumen 1, apéndice F), con la diferencia de que la respuesta va en el mismo archivo y no en el «Disk 12».

```
.d                                          .ds
ELEMENTS                                    ELEMENTS
 K1, 1-GND = FCONVE (H1)    impedancia       KE-K1, 1-GND = H1
 K2, 3-4 = FCONVJ (H2)      admitancia       KJ-K2, 3-4 = H2
FUNCTIONS                                   FUNCTIONS
 TABLE H1                   por tabla        TABLE H1
  0, 2,  0.05, 1.9025, ...                    0, 2,  0.05, 1.9025, ...
 H2 (T) = EXP(-2*T)         por ecuación     H2(1) = exp(-2*x)
```

- **Impedancia** (`FCONVE`, `KE`): la tensión del elemento es la convolución de su corriente, `v(t)` = $\int_0^t$ `h(t−τ)·i(τ) dτ`.
- **Admitancia** (`FCONVJ`, `KJ`): la corriente es la convolución de la tensión.

Una caja de varias puertas se arma, como en SCEPTRE, con un `K` entre cada par de nodos. Por dentro, como SCEPTRE, el elemento es una fuente (la historia, lo ya calculado) en serie con una resistencia (el último tramo, que depende de la corriente desconocida): `v = e + r·i`. siru guarda la historia en cada paso aceptado y calcula `e` y `r` para cada etapa del Runge-Kutta, suponiendo `h` e `i` lineales entre puntos (la fórmula F.9 del manual, exacta para esa suposición):

el error es del orden del cuadrado del paso. Si el circuito no tiene otros estados, el `.din` lleva un estado reloj (`conv_reloj`, derivada 1) para que haya algo que integrar.

Cuidados:

- El **paso adaptativo no ve el error de la convolución** (sólo mira los estados): con un `K` conviene un paso fijo (`SIRU ADAPTIVE STEP = 0` en un `.d`, `SIRU_ADAPT=[0]` en un `.ds`). siru lo avisa al arrancar.
- **Llaves.** Una llave en serie con un `K` de impedancia que se abre, o en paralelo que se abre, anda. Una que lo **cortocircuita** (o que abre la rama de un `K` de admitancia) no: en el comienzo del paso el `K` es una fuente que vale su historia, y la llave le impone algo contra ella, como cortocircuitar un capacitor cargado. Haría falta un impulso; calcula dice que el sistema no se determina y siru agrega esta explicación.
- Un `.din` con elementos `K` no se corre solo (`siru circuito.din`): la historia la guarda siru al correr el `.d` o el `.ds`, y lo dice.
- Por tabla, fuera de la tabla vale el último valor; para un tiempo negativo, 0 (la respuesta es causal). Por ecuación, la función tiene un solo argumento y usa `exp`, `log`, `sqrt`, `sin`, `cos`, `tan`, `abs`, `min`, `max`, `pow` y las hiperbólicas.
- Sólo con Runge-Kutta, no con GLM.
- **El costo.** La convolución tiene memoria, y siru hace la cuenta de dos maneras: - **Recursiva**, si `h` por ecuación es una suma de exponenciales (`exp`, `cos`, `sin`, `cosh`, `sinh` de `a·x`, sumadas y multiplicadas: por ejemplo la impedancia de una caja R, L, C, como en `a06b`). siru lo reconoce solo y no guarda historia: el costo por paso es fijo, y la cuenta es exacta para la corriente lineal entre puntos. - **Directa**, para cualquier otra `h` (por tabla, o una ecuación que no es suma de exponenciales): guarda la historia y cada paso la suma entera. La historia se **comprime** como el `CMPAC` de SCEPTRE: cada `cuenta` pasos se sacan los puntos que sobran (donde la corriente es casi una recta) mientras el área perdida no pase de `criterio` por el área de la historia. Por omisión, cada 50 pasos con criterio 10^{-6}; se cambia con los controles de SCEPTRE, en un `.d` o un `.ds`: `COMPRESSION COUNT = 100` (0 la apaga) y `COMPRESSION CRITERION = 1E-5`. La suma se corta también en los puntos de `h`, como SCEPTRE: con `h` por tabla el costo por paso queda acotado por la tabla y la compresión ahorra tiempo (diez veces con una corriente constante); con `h` por ecuación que no es exponencial ahorra memoria, no tiempo.

Ejemplos, con su prueba en `tests/convolucion.t`: en `siru/corpus/corpus/inventados`, la caja RC escrita (`caja-rc.ds`) y como `K` por ecuación y por tabla (`caja-k-ecuacion.ds`, `caja-k-tabla.ds`); la caja RL escrita (`caja-rl.d`) y como `FCONVJ` (`caja-kj.d`). El plan y las fórmulas: `PLAN_CONVOLUCION.md`.

4.6. 6. Llaves

Una llave es un elemento `SB` (`S1` en un `.d`) cuyo valor es su condición: 1 (o verdadero) cerrada, 0 abierta.

```
S1, 2-3 = T<1                       .d
SB-S1, 2-3 = (T<1)+(T>3)            .ds
S2, 3-GND = VC1>0.5                 por una variable del circuito
```

La condición puede usar cualquier variable del circuito, los parámetros y `T`. Se evalúa **al empezar cada paso**; si cambia en medio de un paso, la llave cambia en el siguiente. Cada combinación de llaves es una configuración, y su sistema se arma la primera vez que se entra en ella (quedan los `.mdin`). Una llave abierta puede dejar nodos sin camino a masa (una isla): se les fija potencial cero y no es error. Ejemplos: `slex/d2datdinc/pruebas/isla.d`, `llave-lazo.ds`, `llave-corte.ds`.

4.6.1. `anterior(X)`: la histéresis

En la condición de una llave, `anterior(X)` es el valor que la expresión `X` tenía al empezar el paso (18/11/2026). `X` puede ser la misma llave, y así se escribe una histéresis: la llave cierra cuando el control pasa un umbral, abre cuando baja de otro menor y entre los dos sigue como estaba.

```
SB-S1, 1-2 = if(anterior(S1)>0.5, VC>0.5, VC>1.5)
```

Al arrancar `anterior(X)` vale 0: esa llave arranca abierta. Es lo que `cir2dsa` escribe de una llave de SPICE con `.model sw (vt= vh=)`: los umbrales son `VT-VH` y `VT+VH`. `anterior` se admite en la condición de una llave y en el valor de una rama, y no pasa a SCEPTRE. En el valor de una rama, `ddt(X)` es la derivada de la expresión `X` respecto del tiempo (la de las fuentes B de SPICE): al terminar cada paso es el cociente de diferencias `(X - anterior(X)) / (T - anterior(T))`, y al empezarlo, o en un paso muy corto, lo que valía antes. Arranca en 0. Prueba: `tests/histeresis_e2e.t` en siru.

4.6.2. 6.1 Llaves o `if(...)`: la diferencia

Las dos eligen entre ecuaciones según una condición. La diferencia es **cuándo** se decide y **qué pasa en el salto**.

	llave (`SB`, `S1`)	`if(c, a, b)` en una ecuación
cuándo se evalúa la condición	una vez, al empezar cada paso (el sistema del instante)	en cada iteración de Newton y en cada etapa del Runge-Kutta
durante el paso	la configuración queda fija: un sistema suave, sin saltos	puede cambiar de rama en medio del paso, o ir y volver entre dos iteraciones
el salto	siru lo resuelve: trasvase de carga o flujo y equilibrio (`-I.mdin`, `-E.mdin`)	no hay tratamiento: los estados siguen como estaban
para qué sirve	cambios de topología, donde los estados pueden saltar	leyes por tramos con los estados continuos
robustez	buena: cada configuración es su propio sistema	más frágil: cerca del cambio, con paso grande, Newton puede oscilar entre las ramas

Un ejemplo de `if` es el cambio de fase (`siru/corpus/corpus/perl/cambiofase.ds`): la entalpía es continua y sólo cambia la ley, calor sensible fuera del intervalo de fusión y latente adentro. Las llaves de hoy son ramas de un circuito, así que un sistema sin circuito, como ese, sólo puede usar `if`.

`if`, `and`, `or`, `not` y `xor` pasan tal cual al `.din` (desde r1899) y Calcula los evalúa con ExprTk. Un estado se puede declarar `TEMP=0 D:DTEMP` en `DEFINED_PARAMETERS` con la derivada dada por una ecuación de `EQUATIONS` (desde r1901), como en `cambiofase.ds`: la temperatura sube hasta la de fusión, se queda ahí mientras funde y después sigue subiendo.

4.7. 7. Lazos de capacitores y cortes de bobinas

Dos capacitores en paralelo, un capacitor en paralelo con una batería, o una malla de capacitores y baterías, es un **lazo**: los voltajes no son independientes. Bobinas en serie en un nodo sin otra salida, o con una fuente de corriente, es un **corte**: las corrientes no son independientes. No hay nada que escribir: `siru` los encuentra, integra con la ley del lazo (o del corte) en derivadas, y

cuando una llave hace aparecer un lazo o un corte con los estados desparejos hace el **equilibrio** —el trasvase instantáneo de carga o de flujo— y escribe una fila más con el mismo `t`. En el `.simul` se ve como un salto.

Al arrancar, si una fuente impone un estado —una bobina en serie con una fuente de corriente, un capacitor en paralelo con una batería— y el archivo no da su condición inicial, el estado arranca con lo que impone la fuente y no hay salto (`arranque: il44 = 10 (lo impone una fuente y el archivo no da su condicion inicial)`); es lo que hace SCEPTRE. Si el archivo la da (`IL44 = 0`), hay equilibrio como en cualquier otro caso: la bobina salta a 10 y las acopladas con ella reciben el golpe.

Una fuente que varía adentro de un lazo o un corte entra con su derivada: la del modelo si la trae (`EV`, `ESIN`), y si no la numérica. En un lazo la capacidad puede ser constante o depender del voltaje de capacitores:

- del **propio** (`C1, 2-GND = 1+0.5*VC1`);
- del de **otro capacitor** (`C2, 1-2 = X1(1.+0.2*VC1**2)`; `CC2 = f(VCC1)` en el transistor JCNTRAN de `d04.d`);
- de la **corriente de una fuente** cuyo valor es función de un voltaje en paralelo con un capacitor (`CE = f(JE)`, con `JE` el diodo de la juntura: siru cambia `JE` por su ecuación y el voltaje del diodo por el de `CE`).

Si depende de otra cosa (el tiempo, una corriente cualquiera), en un lazo es un error que lo dice. Ejemplos: `siru/corpus/corpus/llaves/*.d`, `slex/d2datdinc/pruebas/capacidad-variable.d`, `slex/d2datdinc/pruebas/sceptre/lazo-c-ajeno.d`.

El estado de un capacitor es su carga, `Q = C·V`, y la corriente es `dQ/dt`: con una capacidad variable eso incluye el término `V·dC/dt`. SCEPTRE hace otra cosa: integra `I = C·dV/dt`, con `C` evaluada en el punto (la capacidad incremental). Con capacidad constante es lo mismo; con capacidad variable **los resultados difieren** y no es un error de ninguno de los dos: en `slex/d2datdinc/pruebas/sceptre/c-variable.d` (`C1 = 1 + 0.2·VC1`^{2} cargado a 10 V por 1 Ω) SCEPTRE da `VC1 = 6.17` en `t = 4` y siru `4.74`. Para escribir en siru la capacidad incremental de SCEPTRE `c(V)` hay que dar `C = (1/V)·∫c dV` (para `c = 1 + 0.2·V`^{2}, `C = 1 + 0.2·V`^{2}`/3`): con eso siru da lo mismo que SCEPTRE (`c-variable-q.d`: 6.166829 contra 6.166827).

Con lazos o cortes, el punto de operación (11 bis) se resuelve en un sistema aparte, `NOMBRE-[bits]D.mdin`: el instante con las ecuaciones originales en lugar de las leyes derivadas, que con las derivadas en cero no atan nada.

4.8. 8. Acoplamientos magnéticos

Inductancia mutua, como en SCEPTRE: una línea `M, L1-L2 = valor` entre dos bobinas `LB` que ya existen. Agrega `M·dI2/dt` a la tensión de `L1` y viceversa. Si una llave abre la rama de una de las dos, la otra recibe el golpe: `ΔI2 = (M/L2)·I1` (se conserva el flujo concatenado).

```
L1, 2-GND = 1
L2, 3-GND = 4
M,  L1-L2 = 1.8
```

Transformador con núcleo: cada devanado es un `MW` con sus vueltas y la derivada del flujo del núcleo; el flujo es un estado propio y la fuerza magnetomotriz una ecuación propia (sección 10).

```
ELEMENTS
    MW1, 2-GND = 1, DPHI1          N1 = 1 vuelta
    MW2, 3-GND = 2, DPHI1          N2 = 2 vueltas
VARIABLES
    DPHI1 = 0
```

```
EQUATIONS
    1*IMW1 + 2*IMW2 - REL*PHI1     N1·I1 + N2·I2 = reluctancia · flujo
DEFINED PARAMETERS
    REL = 1
    PHI1 = 0
```

REL puede ser una expresión (un núcleo que satura). Ejemplos: `slex/d2datdinc/pruebas/mutua.d`, `nucleo.d`, `siru/corpus/.../trafo*.d`.

4.9. 9. Funciones y tablas

FUNCTIONS: `NOMBRE(args) = expresión`. En un `.d` los argumentos llevan nombre. En un `.ds` se puede declarar la cantidad y llamarlos `x`, `y`, `z`...; también se admite la lista explícita de nombres de §3.5.

```
FUNCTIONS                                .d
QC(PC,V) = PC*(1+0.1*V)

FUNCTIONS                                .ds
QC(2) = x*(1+0.1*y)
```

Hasta 6 argumentos las compila Calcula; con más, `d2datdinc` reemplaza cada llamada por el cuerpo de la función, así que andan igual. Las constantes de `CONSTANTS` se pueden usar adentro.

Tablas: una función de una variable dada por puntos, interpolada (lineal; un `x` repetido es un salto). Si la tabla se evalúa en el tiempo, como `TABLE 1(TIME)`, cada salto es un punto de quiebre de la corrida. siru corta el paso justo antes del salto y lo cruza con un paso mínimo, así ningún paso promedia el valor de antes con el de después. La salida muestra el salto en dos filas, una a cada lado (Integrador, r2104; `siru/corpus/EXACTAS.md`). Vale también para una tabla en archivo aparte, y para funciones que saltan en el tiempo, aunque el tiempo esté transformado de forma lineal: `if(T<1,...)`, `if(4*T-2 < 2,...)`, `T < TCORTE` con un parámetro, `sgn(T-1)`, `mod(T,2)`, `T% 3`, `floor(T/2)`, una tabla evaluada en `2*T+1`, o una función propia evaluada en `T` que haga eso. Vale con Runge-Kutta, con GLM y con PGLM. En un método multipaso, como BDF2, el paso que cruza el salto y el siguiente se dan como el primero de la corrida. Lo que no se reconoce, como `sin(T) > 0.5` o un salto de una variable del circuito, pierde medio paso en cada salto. Ejemplos: `siru/corpus/corpus/inventados/salto-*.ds`.

```
FUNCTIONS                                .d: adentro del archivo
TABLE 1
  0,0, 1,1, 2,0
ELEMENTS
 E1, GND-1 = TABLE 1 (T)                o T1(T), o XTABLE(T1, VC1)

ELEMENTS                                 .ds: en un archivo de dos columnas
 EB-E1, GND-1 = Table('tabla.txt','TIEMPO','DATO',T,1)
```

Desde el 02/10/2026 un `.ds` también acepta la forma de SCEPTRE, con la tabla adentro del archivo: en `FUNCTIONS TABLE 1` y debajo los pares (o `TABLE 1, 0,0, 1,1, 2,0` en una línea), y en el valor `TABLE 1 (T)` o `TABLE 1(TIME)`. La forma corta `T1(T)` en un `.ds` sólo vale si la tabla ya se leyó (antes de `ELEMENTS`); si no, se toma por una función. Ejemplo: `siru/corpus/.../needsderiJ.ds`, que da lo mismo que su `.d`.

Sección `TABLE` general (lectura textual implementada): en `.d` y `.ds` se puede incorporar una tabla al mismo documento que `RESULTADOS` o `PARAPLOT`. Slex guarda las líneas de la tabla, incluidos comentarios de cabecera y cifras, en `GLOBAL.tablas` del DAT sin redondearlas. La sección termina al empezar otra sección reconocida, en `END` o en EOF; no requiere `END TABLE`, `DOCUMENT`, nombre de archivo ni extensión. `TABLE` o `TABLE FORMATO=CSV` usan el nombre lógico `SIMULADO`. Para varias fuentes, usar nombres distintos: `TABLE MEDIDAS FORMATO=CSV`, `TABLE`

`CALCULO FORMATO=SIMUL`. Los formatos textuales declarables son `SIMUL`, `CSV` y `TEXTO`; sin declaración se usa `SIMUL`. En `.d`, `TABLE NOMBRE` sin `FORMATO=` sigue siendo la tabla histórica de pares de SCEPTRE; en `.ds`, un nombre alfabético sin formato es general. El lector no interpreta las columnas: esa tarea corresponde a Tabular2. El adaptador DAT→Tabular2 y el consumo gráfico de tablas incorporadas aún faltan; por tanto el siguiente ejemplo prueba la lectura al DAT, no el dibujo automático:

```
TABLE
# T A B
0 20 18
1 22 19
RESULTADOS
A,B
```

Una entrada estándar con esas secciones seguirá siendo un único `.d` o `.ds`; el programa deberá recibir el dialecto por opción, porque stdin no tiene extensión. `END` conserva su sentido histórico de fin de fuente: si se usa, va después de todas las secciones que se quieran leer. Siru, Simplot y los demás consumidores deben compartir el dato producido por el mismo lector, sin una segunda gramática de tablas.

Para Simplot, PARAPLOT es una sección opcional del `.ds`, no un tipo de archivo independiente. El contrato de entrada comprende las siguientes organizaciones; no implica que todos los consumidores hayan implementado ya cada recorrido:

Organización	Descripción
`.ds` autocontenido	Una o más secciones `TABLE`, con PARAPLOT para el pedido explícito o sin PARAPLOT para el dibujo automático.
`.ds` y tablas externas	Uno o más archivos de tablas, identificados por el documento; la fuente homónima puede ser implícita.
Entrada estándar	El mismo documento textual con todas sus tablas incorporadas, antes o después de PARAPLOT. No requiere `DOCUMENT` ni `END`.

Los ejemplos ejecutables están en `simplot/corpus/entradas-ds/`. Las pruebas recorren Slex → DAT → Tabular2 → Simplot con archivos separados, tablas incorporadas y stdin textual. El protocolo JSON anterior sigue disponible por compatibilidad, pero el pipe actual de Siru utiliza `TABLE` y PARAPLOT en texto.

Contenido binario incorporado (pendiente): un `.d` o `.ds` sigue siendo texto; por eso una tabla binaria deberá codificarse, por ejemplo en base64, dentro de `TABLE`. El DAT debe indicar por separado el nombre lógico de la fuente, el formato original, la codificación y el contenido; no debe presentar el texto base64 como si fueran filas numéricas. El adaptador decodificará los bytes y los entregará al lector de Tabular2 con su longitud explícita. Ese lector especializado, no el lexer de Slex, interpretará POS4 u otro formato binario. Falta definir y probar la sintaxis exacta de la declaración y el delimitador del bloque codificado; este párrafo no habilita aún ninguna entrada binaria en `.ds`.

Nombre de la tabla producida: al ejecutar `NOMBRE.ds`, Siru emite `TABLE NOMBRE FORMATO=SIMUL` mediante `--documento`, sin ruta ni extensión. No toma el nombre de un archivo de salida alternativo, ni agrega sus sufijos: `caso.ds` sigue identificando la tabla `caso` aunque la salida se guarde como `caso-1.simul`. El pipe actual a Simplot transmite ese mismo nombre en el campo `tabla` de su cabecera. Los nombres conservan mayúsculas, guiones y puntos; con `FORMATO=` también pueden comenzar por un número. La referencia implícita histórica `SIMULADO`

de PARAPLOT se conserva por compatibilidad. La tabla numérica `.simul` sigue en su formato habitual; `TABLE NOMBRE` es el encabezado de sección al incorporarla a un documento.

Estado del recorrido: Slex produce `GLOBAL.tablas` y vincula las fuentes gráficas con sus tablas incorporadas. Simplot materializa privadamente esos datos en su formato declarado y los carga mediante Tabular2; no implementa otra interpolación ni otro lector numérico. Siru emite la fuente con su tabla nombrada mediante `--documento` y transmite a Simplot un documento gráfico textual. Las tablas SCEPTRE de pares siguen por su ruta histórica. El binario codificado dentro de `.ds` continúa pendiente; los archivos externos se pueden normalizar con Tabular2 antes de incorporarlos.

El archivo `tabla.txt` es de tabular2: una línea `#TIEMPO DATO` y las filas; su ruta es relativa al `.ds` (o al directorio de arriba, como corría Siru). Las `x` tienen que ir creciendo. `Table()` también se puede usar adentro de una función. Ejemplos: `slex/d2datdinc/pruebas/tablas.d`, `siru/corpus/.../tabla.ds`.

Funciones en Fortran: `EXTERNAL`. Una función escrita en Fortran, en un `.f90`, `.f` o `.for` aparte (la "External Function" del manual de SCEPTRE, 2.2.2 y 2.7), se declara en `FUNCTIONS` con su nombre, su cantidad de argumentos (de 1 a 12) y la ruta del fuente, relativa al `.d` o al `.ds`:

```
ELEMENTS                                .d
 E1, GND-1 = FGEN (0., 5., 20., 40., 100., 100., 300., 2., TIME)
 L2, 2-GND = FCOR (IL2, IR1, 25., 100.)
FUNCTIONS
 EXTERNAL FGEN (9) fgen.f90
 EXTERNAL FCOR (4) fcor.for
```

El fuente tiene que ser una `FUNCTION` de doble precisión, con los argumentos por referencia, que es lo normal en Fortran:

```
      FUNCTION FCOR(IL, IR, L0, ISAT)
      DOUBLE PRECISION FCOR, IL, IR, L0, ISAT
      FCOR = L0 / (1D0 + ((IL + IR)/ISAT)**2)
      END FUNCTION
```

`d2datdinc` lleva la declaración al `.din` (sección `externas`, con la ruta absoluta); Calcula compila el fuente con `gfortran -shared -fPIC -O2` a un `.so` en `~/.cache/calcula/externas`, lo vuelve a compilar si el fuente cambia, lo carga y llama a `fgen_`, el nombre en minúsculas con el `_` que le pone gfortran. También acepta un `.so` ya compilado como ruta. Hace falta `gfortran` donde se corre, y **`CALCULA_EXTERNAS=1` en el entorno**: es código de afuera, y Calcula no lo carga sin ese permiso. Por eso Puyuspa, que corre archivos de cualquiera, no ejecuta un ejemplo con funciones externas salvo que se lo instale en el servidor. Los mensajes de error están en la sección 13. Ejemplo: `siru/corpus/corpus/fortran/subprograma.d`, con `fgen.f90` y `fcor.f90`.

`ETABLE`: una tabla en un archivo aparte, por número. Es la forma de SCEPTRE y Simusol: `ETABLE(N, X, Y, V, M)` es la tabla número `N`, interpolada en `V`, con las abscisas en la columna `X` y los valores en la columna `Y` (contadas desde 0) y el método `M` (1 lineal, 2 spline). Qué archivo es cada número lo dice `etable.txt`, junto al archivo, con una línea `número archivo` por tabla:

```
DEFINED PARAMETERS                      con_FOR.d
PP = X1(ETABLE(2,0,2,TIME,2))

2 con_FOR.etable                        etable.txt
```

siru la tiene de dos maneras. **La normal**: no hay que declarar nada; libdin la traduce a `tabla('con_FOR.etable','0','2',t,2)` de Calcula (desde calcula 6.0-34, que acepta las columnas por número). **La de Fortran**: si el archivo la declara `EXTERNAL ETABLE (5) RUTA/tabular2/fortran/etable.f90`, se llama a esa función —la misma

que enlaza SCEPTRE—, con `CALCULA_EXTERNAS=1`. Las dos dan lo mismo. Ejemplos: `siru/corpus/corpus/fortran/con_FOR_1.d` (la normal) y `con_FOR_1.ds` (la de Fortran).

Un archivo sin circuito ni estados que pide una corrida —una tabla o una función a lo largo del tiempo, como esos— se resuelve punto por punto, con el paso de `SIRU_PASO`, o el de `MAXIMUM STEP SIZE` en un `.d`, o la milésima parte del tiempo final (11); la salida lleva el tiempo y los `OUTPUTS`.

4.10. 10. Ecuaciones propias, estados propios, parámetros

Un circuito puede traer ecuaciones además de las suyas, y un archivo puede no tener circuito y ser sólo ecuaciones.

- **`EQUATIONS`**: cada línea es una expresión igualada a cero. Puede nombrar las variables del circuito (`VC1`, `IMW1`), los parámetros, `T`, `DT`, `TICK`.
- **`VARIABLES`**: las incógnitas que esas ecuaciones agregan, con su semilla: `Z=0`. Una que se llame `D<X>` con `X` definida en `DEFINED PARAMETERS` es la derivada de `X`, y `X` pasa a ser un **estado** que se integra. Cada ecuación propia necesita una incógnita propia: si no cierran los grados de libertad, `siru` lo dice.
- **`DEFINED PARAMETERS`**: `P = 22` es un parámetro; `P = expresión` es una incógnita propia con su ecuación (`PDIS = PX1*2`); `DPX1 = PX2` junto con `PX1 = 1` hace de `PX1` un estado con derivada `PX2` (la mecánica de `a13.d`). También van las condiciones iniciales (`QC1 = 0`, `IL1 = 0`) y, en un `.ds`, `T=0` y `TIEMPOFINAL`.
- **`CONSTANTS`**: `K=1e-3`, números con nombre para usar en expresiones y funciones.

Un sistema sin circuito:

```
CIRCUIT_DESCRIPTION
  y'' = -y
VARIABLES
DY1=0
DY2=0
EQUATIONS
DY1-Y2
DY2+Y1
OUTPUTS
T:Y1,Y2
DEFINED_PARAMETERS
T=0
TIEMPOFINAL=3
Y1=1
Y2=0
RUN_CONTROLS
INTEGRA=[112]
CONTROL_EQUATIONS
T<TIEMPOFINAL
END
```

Ejemplos: `slex/d2datdinc/pruebas/propias.d`, `parametros-propios.d`, `nucleo.d`, `nulo.d`.

4.11. 11. La corrida

Selector único INTEGRA Desde el 29/10/2026 se usa INTEGRA=[...] en .d, .ds y .dsa. Reemplaza a RK, GLM, ROS, PGLM y SPLIT como selectores públicos; los motores internos y sus descriptores conservan sus nombres. No se pueden combinar selectores antiguos con INTEGRA.

Números globales	Métodos
-1	Un punto, sin integración
1	Euler explícito
2, 3	ROS2, ROS3
4, 5	Euler IMEX, ARS222
6, 7	Lie–Trotter, Strang
20, 21	BDF2 efectivo, BDF2 directo
22, 23, 24	BDF3, BDF4, BDF5
25, 26	BDF de orden variable, con historia directa o Nordsieck
30	Adams–Bashforth 2
101–107, 110–118	Tablas antes numeradas en GLM, sin alterar su numeración

INTEGRA=[21,30] alterna BDF2 y AB2 por pasos aceptados y vuelve al primero; un rechazo no avanza la lista. También se pueden alternar motores diferentes: INTEGRA=[2,21,30] combina ROS2, BDF2 y Adams, conservando la historia física aceptada. IMEX y composición requieren sus particiones o mapas y paso fijo. La coordinación de llaves también está disponible para ROS, IMEX y composición: se decide entre pasos, se ajustan lazos/cortes y se fija la conexión durante las etapas y reintentos. No se buscan conmutaciones interiores. La convolución conserva sus restricciones. Catálogo, compatibilidad y límites en INTEGRA (`../../../siru/doc/INTEGRA.md`).

4.11.1. Composición de subflujos (SPLIT)

Se declara cómo avanzar cada subproblema, no solamente su derivada. Para el oscilador `Q'=P`, `P'=-Q`, A mueve Q manteniendo P y B mueve P manteniendo Q. En `RUN_CONTROLS` de un `.ds`:

```
INTEGRA=[7]
SPLIT_AUTONOMO=[1]
SPLIT_EXACTOS=[1]
SPLIT_A=[Q,Q+SPLIT_H*P,P,P]
SPLIT_B=[Q,Q,P,P-SPLIT_H*Q]
SIRU_PASO=[0.1]
SIRU_ADAPT=[0]
```

Selector	Orden de ejecución de subavances	Orden esperado con subflujos exactos suaves
`INTEGRA=[6]`	A(h), luego B(h)	1 (Lie–Trotter)
`INTEGRA=[7]`	A(h/2), luego B(h), luego A(h/2)	2 (Strang)
`INTEGRA=[227]`	Todos los flujos en el orden de `SPLIT_FLUJOS`, cada uno con h	1 (Lie–Trotter-N)
`INTEGRA=[228]`	Mitad de paso para cada flujo salvo el último, paso completo al último y recorrido inverso de las mitades	2 (Strang-N)

Cada lista debe incluir una expresión para **todos** los estados, incluso los que se conservan. Las expresiones de un mismo mapa se evalúan simultáneamente sobre su estado de entrada. `SPLIT_H`

es la duración completa de avance del estado en ese subavance. En modelos no autónomos de dos flujos, `SPLIT_TIEMPO_A/B` indican cuánto cambia el reloj por unidad de duración de A o B; para N flujos se declara uno por mapa. En ambos casos, los pesos suman uno. Se permite aritmética `+ - * / ^` y paréntesis con estados, parámetros y el reloj. No se admiten derivadas, variables algebraicas, funciones o tablas en estos mapas.

Para los selectores 227/228 se declara `SPLIT_FLUJOS=[A,B,C,...]` (entre 2 y 16 nombres únicos de A a P) y un mapa completo por flujo: `SPLIT_A`, `SPLIT_B`, etc. Lie–Trotter-N recorre todos en el orden de la lista. Strang-N recorre medias duraciones en ese orden hasta el penúltimo, una duración completa en el último y vuelve por las medias duraciones en orden inverso. Los mapas deben asignar expresión a todos los estados. En modelos no autónomos se declara un `SPLIT_TIEMPO_*` por flujo y sus valores deben sumar uno; por ejemplo, para A/B/C pueden ser 0.3, 0.3 y 0.4. El reloj cambia según el peso del flujo por `SPLIT_H`; el estado avanza durante `SPLIT_H` completo.

`SPLIT_AUTONOMO=[1]` y `SPLIT_EXACTOS=[1]` declaran mapas autónomos y exactos. Para los métodos de dos flujos no autónomos se usan `SPLIT_TIEMPO_A` y `SPLIT_TIEMPO_B`; para N, un `SPLIT_TIEMPO_*` por flujo. El autor declara que cada mapa es el flujo exacto de su parte y que vale la identidad con duración cero. El programa verifica forma, nombres y capacidades, **no demuestra esas propiedades matemáticas**. Un mapa de Euler no es exacto: debe declararse con `SPLIT_EXACTOS=[0]` y su orden. El modelo residual completo se conserva y Calcula lo resuelve en los estados intermedios; debe ser válido allí.

También se admiten mapas aproximados con `SPLIT_EXACTOS=[0]`. En ese caso es obligatorio indicar `SPLIT_ORDEN_A=[p]`, etc., para cada flujo. Cada mapa aplica una actualización numérica por subavance; no se ejecuta un integrador interno. El orden efectivo que se usa para la adaptación es el mínimo entre el orden de la composición y los órdenes declarados. El autor debe justificar esos órdenes: el programa no los demuestra. El ejemplo oscilador-tres-flujos-aprox.M.dsa (`../../../corpus/corpus/SPLIT/oscilador-tres-flujos-aprox.M.dsa`, con su `.DU.dsa` al lado; era `oscilador-tres-flujos-aprox.ds`) usa Strang-N con un mapa de Euler de orden uno; por eso el orden efectivo es uno y `SIRU_ORDEN_ERROR` debe ser dos.

En `.d` se escriben `SPLIT FLUJOS`, `SPLIT AUTONOMO`, `SPLIT EXACTOS`, `SPLIT ORDEN A`, `SPLIT A`, `SPLIT B`, etc. (con espacios), junto con `STARTING STEP SIZE` y `SIRU ADAPTIVE STEP = 0` para paso fijo. lc-strang.d (`../../../corpus/corpus/SPLIT/lc-strang.d`) muestra dos flujos; lc-strang-tres-flujos.d (`../../../corpus/corpus/SPLIT/lc-strang-tres-flujos.d`) muestra Strang de tres. Ambos declaran mapas de **QC1**, la carga que libdin usa como estado, aunque piden **VC1** como salida; C=L=1. También están oscilador-1.M.dsa (`../../../corpus/corpus/SPLIT/oscilador-1.M.dsa`, con su `.DU.dsa` al lado; era `oscilador-1.ds`), oscilador-2.M.dsa (`../../../corpus/corpus/SPLIT/oscilador-2.M.dsa`, con su `.DU.dsa` al lado; era `oscilador-2.ds`), oscilador-tres-flujos.M.dsa (`../../../corpus/corpus/SPLIT/oscilador-tres-flujos.M.dsa`, con su `.DU.dsa` al lado; era `oscilador-tres-flujos.ds`) y oscilador-tres-flujos-aprox.M.dsa (`../../../corpus/corpus/SPLIT/oscilador-tres-flujos-aprox.M.dsa`, con su `.DU.dsa` al lado; era `oscilador-tres-flujos-aprox.ds`).

La cadena es `.d/.ds` → `d2datdinc --din` → `NOMBRE-de-d2datdinc-SPLIT*.din` → `siru`. Los mapas exactos usan descriptores v1–v5 según el selector; cualquier composición con mapas aproximados genera v6 con los órdenes declarados, la lista de flujos, secuencia y pesos del reloj. Todos admiten paso fijo/adaptativo y tiempo autónomo/no autónomo. No requiere fuente, `.dat` ni catálogo. `split_h` y `split_flujo` son datos internos de Calcula, `split_p_estado` es la salida del mapa y `p_estado` es la salida virtual del paso completo. Los prefijos `split_` y `p_` están reservados.

Solo se confirma el reloj y se escribe una fila al terminar toda la secuencia. Los tiempos internos no se suman: la secuencia completa de cada método representa un único paso físico h. Un fallo de subflujo aborta sin publicar trayectoria parcial. El informe cuenta pasos completos,

subavances y consultas; se aceptan último paso acortado y `--paso-factor`.

Límites: un selector y mapas exactos o aproximados con orden declarado, sin eventos interiores, convolución, DT/TICK, DC/AC, RERUN/CONTINUE, funciones propias o externas ni tablas internas o externas. No se ofrece integrador interno ni tolerancia independiente por flujo; la adaptación usa duplicación de la composición completa. Yoshida4/6 pueden requerir duraciones negativas; Lie–Trotter-N y Strang-N no. El núcleo genérico es `Siru::Composicion`, y `Siru::SPLIT` lo conecta con el conductor transaccional y Calcula. Pruebas: `prove -Ilib tests/split_nucleo.t tests/split.t`.

4.11.2. GLM fijo o adaptativo

`INTEGRA=[1]` selecciona Euler explícito, `[20]` BDF2 efectivo, `[21]` BDF2 directo y `[30]` Adams–Bashforth 2. Son identificadores globales que conservan la numeración GLM. Se admiten listas compatibles, como INTEGRA=[21,30]; las condiciones de historia, adaptación, topología y convolución dependen del método y se detallan en el contrato GLM y sus documentos asociados. El artefacto generado contiene el descriptor y se ejecuta sin fuente ni `.dat`.

En `RUN_CONTROLS` de un `.ds`, por ejemplo:

```
INTEGRA=[21]
SIRU_ADAPT=[1]
SIRU_PASO=[0.1]
SIRU_PASO_MIN=[1e-8]
SIRU_PASO_MAX=[0.2]
SIRU_ABSOLUTO=[1e-5]
SIRU_RELATIVO=[0]
SIRU_ORDEN_ERROR=[2]
```

En `.d` se usan los controles con espacios de la tabla de abajo. `[0]` en `SIRU_ADAPT` conserva el modo fijo. En adaptación, BDF2/AB2 requieren orden del estimador 2 y se comparan con Euler; AB3–AB5 requieren sus órdenes 3–5 y se comparan con el Adams inmediatamente inferior. Los arranques Kutta3 y RK4 de AB3–AB5 usan dos medios pasos y órdenes de control 4 o 5. Se acepta siempre el candidato principal. Son políticas conservadoras, no una garantía de error global ni BDF de orden variable. Un fallo de Calcula aborta; el rechazo por tolerancia no mueve historia ni tiempo. No se admite `DT/TICK` dentro del modelo adaptativo.

Los `.din` nuevos usan contrato GLM v2; los v1 archivados conservan el paso fijo y deben regenerarse para activar adaptación. `estadisticas ultima integracion:` en el `.informe` registra intentos, aceptaciones, descartes y consultas del conductor, incluyendo las de comparación. En RERUN/CONTINUE de otros motores describe la última integración, no la suma de corridas. ROS conserva su registro `estadisticas ROS:`. Fórmulas, límites y ejemplos en GLM_CONTRATO.md (`../../../siru/doc/GLM_CONTRATO.md`).

4.11.3. Paso por error de truncamiento (el de SPICE)

Con la regla del trapecio, `INTEGRA=[112]`, el control `TRTOL` pide que el paso lo decida el error local de truncamiento, como en SPICE (tomado de ngspice 47, 18/11/2026). El error se estima con los últimos puntos aceptados, sin resolver nada de más. En `CONTROLES`:

```
INTEGRA=[112]
SIRU_PASO=[1e-5]
SIRU_PASO_MAX=[1e-5]
SIRU_RELATIVO=[1e-3]
TRTOL=[7]
CHGTOL=[1e-14]
ABSTOL=[1e-12]
```

- `TRTOL`: el factor por el que se multiplica la tolerancia (`trtol` de SPICE, 7 por omisión). Su presencia elige este control del paso.
- `CHGTOL`: la tolerancia absoluta de cada estado —la carga de un condensador, el flujo de una bobina— (`chgtol`, 1e-14 por omisión).
- `ABSTOL`: la tolerancia absoluta de la derivada del estado —la corriente, la tensión— (`abstol`, 1e-12 por omisión).
- `SIRU_RELATIVO`: la tolerancia relativa (`reltol`).

Un paso se acepta si el que el error tolera supera 0.9 veces el dado; el siguiente no crece más del doble. Al empezar y al cruzar un punto de quiebre de una fuente el paso arranca en la décima parte. `CHGTOL` y `ABSTOL` sin `TRTOL` se rechazan; con `TRTOL` no se admite `SIRU_ABSOLUTO` ni otro método que el 112, y todavía no se combina con convoluciones. Es lo que escribe `cir2dsa` al traducir un `.tran`, con los valores del `.options` o los que SPICE toma por omisión. Prueba: `prove tests/truncamiento.t tests/truncamiento_e2e.t` en siru.

4.11.4. Desde dónde se escribe la tabla (`SALIDA_DESDE`)

`SALIDA_DESDE=[t]` hace que la tabla de resultados empiece en el instante `t` (18/11/2026): las filas anteriores no se escriben. La corrida se hace igual desde el principio, y las medidas se calculan sobre la tabla que queda. Es el `tstart` de un `.tran` de SPICE, y `cir2dsa` lo escribe de ahí.

4.11.5. Partición explícita e implícita (PGLM / IMEX)

Selector	Método	Orden
`INTEGRA=[4]`	Euler IMEX	1
`INTEGRA=[5]`	ARS(2,2,2)	2

La partición se declara en `RUN_CONTROLS`, junto con un paso fijo positivo:

```
INTEGRA=[5]
IMEX_EXPLICITA=[X,-X]
SIRU_PASO=[0.01]
SIRU_ADAPT=[0]
```

Para `X'=-11*X`, la parte explícita anterior es `-X`; la implícita queda definida como el campo completo menos la parte explícita, en este caso `-10*X`. No son dos corridas independientes. `IMEX_EXPLICITA` lleva pares estado,expresión para **todos** los estados reales; en un capacitor CB el estado es QC1, no VC1. Consultarlos con `d2datdinc --topologia`.

Regla	Alcance
Expresiones de la partición	Números finitos, estados, parámetros y tiempo, operadores `+ - * / ^` y paréntesis.
Expresiones excluidas	Funciones, comas interiores, variables algebraicas, derivadas y DT. Las ecuaciones del modelo completo conservan su propio lenguaje.
Selección	INTEGRA=[4] o INTEGRA=[5], con un solo ID; no combinar selectores.

Regla	Alcance
Paso	Fijo, con último paso acortado; no adaptación ni listas de métodos.
Modelo	Campo diferencial determinado por tiempo y estados; no se garantiza resolver DAE arbitrarias ni de índice alto.
Coordinación	Llaves y ajustes de lazos/cortes entre pasos; conexión fija en las etapas.
Combinaciones no admitidas	Convoluciones, DT y RERUN.
Forma `.d`	`IMEX EXPLICITA = [X,-X]`, `STARTING STEP SIZE = 0.01`, `SIRU ADAPTIVE STEP = 0`.

El `.din` generado conserva el descriptor y las tablas, de modo que puede ejecutarse sin la fuente ni el catálogo. La separación explícita conserva sus restricciones de estabilidad. El contrato completo y los límites están en IMEX (`../../../siru/doc/IMEX.md`) y ARS222 (`../../../siru/doc/ARS222.md`).

4.11.6. Rosenbrock (extension ROS)

`INTEGRA=[2]` en RUN CONTROLS (`.d`) o RUN_CONTROLS (`.ds`) selecciona ROS2 de orden dos; el catálogo también ofrece ROS3 `[3]`, ROS3P `[8]`, ROS4 `[9]`, RODAS4 `[154]`, RODAS3 `[155]` y RODAS3.1 `[156]`. No se combina con selectores antiguos ni con IMEX_EXPLICITA. `SIRU ADAPTIVE STEP = 0` en `.d` o `SIRU_ADAPT=[0]` en `.ds` selecciona paso fijo; con 1 usa el estimador embebido del método elegido, las tolerancias SIRU y los límites de paso existentes. RODAS3.1 solo admite paso fijo porque no se verificó el orden del estimador publicado. Los órdenes de los otros estimadores son 2 para ROS2, 3 para ROS3/ROS3P/RODAS3 y 4 para ROS4/RODAS4. Con llaves o lazos/cortes, Siru coordina la conexión y los ajustes al comenzar cada paso, incluidas las paradas adicionales de ese circuito. No verifica llaves dentro del paso. Siguen sin ampliarse convoluciones, DT/TICK, análisis DC, sensibilidad, Monte Carlo, peor caso, RERUN ni CONTINUE; las paradas adicionales e inicialización algebraica del camino sin coordinación conservan sus restricciones.

El archivo generado termina en `-ROS<ID>.din` y se puede ejecutar sin su fuente; incluye OUTPUTS si se pidieron. El motor consulta el campo diferencial en Calcula, calcula un Jacobiano por diferencias y resuelve las etapas lineales. No usa la numeracion RK ni supone que el Jacobiano de Newton sea el de la ecuacion diferencial. Detalles y precauciones de escalas y precision en ROS.md (`../../../siru/doc/ROS.md`).

Los din nuevos usan descriptor ROS v2 fijo o v3 adaptativo, con escalas configurables y comprobacion de diferencias a dos perturbaciones; los v1 archivados conservan el comportamiento anterior. En .ds: `ROS_ESCALAS=[X,1e-6,Z,1e6]`, `ROS_ESCALA_TIEMPO=[1]`, `ROS_DELTA=[1e-4]`, `ROS_CONTROL_JACOBIANO=[1e-6,1e-3,3]`. En .d los nombres llevan espacios en lugar de guiones bajos. Estas opciones controlan la diferenciacion numerica, no activan adaptacion temporal ni cambian la precision interna de Calcula. v3 agrega la seccion adaptacion y los errores virtuales e_estado; solo publica pasos aceptados. Fallos del campo/Jacobiano abortan, no se ocultan reduciendo el paso temporal.

Con adaptacion activa, `ROS_TOLERANCIAS=[X,1e-11,1e-4,Z,10,1e-4]` declara grupos estado, absoluta, relativa; en .d se llama `ROS TOLERANCIAS`. Los estados omitidos usan las tolerancias globales. Se guarda el mapa completo en descriptor v4, de modo que el din sigue siendo autosuficiente. No confundir estas tolerancias temporales con ROS_ESCALAS. El informe `.informe` incluye `estadisticas ROS:` en JSON: pasos aceptados/rechazados, intentos, consultas a Calcula y estado de finalizacion; las consultas del campo y de OUTPUTS se cuentan por separado. Detalles en ROS.md (`../../../siru/doc/ROS.md`).

Para ROS tambien se puede elegir `ROS_JACOBIANO=[SENSIBILIDADES]` (en .d, `ROS JACOBIANO`): genera descriptor v5 y requiere un Calcula con capacidad SENS1. Este resuelve el punto y obtiene el Jacobiano reducido mediante las sensibilidades de los residuos, sin resolver otro punto por cada perturbacion. `ROS_ESCALAS_RESIDUO=[Z,1]` permite escalas de cualquier variable, incluidas algebraicas. Es optativo: los archivos anteriores mantienen campo_central. Funciona con paso fijo o adaptativo; sus fallos abortan sin cambio silencioso de proveedor. Limites y controles numericos en ROS.md (`../../../siru/doc/ROS.md#sensibilidades-del-residuo-v5`).

4.11.7. Controles generales y Runge-Kutta

qué	.d	.ds
hasta cuándo	`STOP TIME = 5`, o `T>5`	`TIEMPOFINAL=5` y `T<TIEMPOFINAL` en `CONTROL_EQUATIONS`
desde cuándo	`START TIME = 0` (0 si no)	`T=0` en `DEFINED_PARAMETERS`
otra parada	`TICK>1000`, `VC1>0.9` en `RUN CONTROLS`	`TICK<=1000`, `VC1<=0.9` en `CONTROL_EQUATIONS` (se sigue mientras vale)
método	`INTEGRATION ROUTINE = TRAP` (sólo `TRAP`, `IMPLICIT`, `XPO`, `RUK`); XPO se traduce a `INTEGRA=[216]`.	`INTEGRA=[112]`, cualquiera de la tabla
varios métodos, rotando	no	`INTEGRA=[118,118,111]`
paso	`STARTING STEP SIZE = 0.05`	`STARTING STEP SIZE = 0.05` (desde r1890; también `MINIMUM` y `MAXIMUM`), o `SIRU_PASO=[0.05]`
semilla de calcula	`SEMILLA VL2 = 0.1`	`SEMILLA VL2 = 0.1`, o `VL2=0.1` en `VARIABLES` si es de una rama
paso adaptativo	`MINIMUM/MAXIMUM STEP SIZE`, `SIRU ABSOLUTE TOLERANCE`, `SIRU RELATIVE TOLERANCE`, `SIRU ERROR ORDER`, `SIRU ADAPTIVE STEP = 1`	`SIRU_PASO_MIN=[..]`, `_PASO_MAX`, `_ABSOLUTO`, `_RELATIVO`, `_ORDEN_ERROR`, `SIRU_ADAPT=[1]`
qué sale	`OUTPUTS` con `VC1, IR1`	igual, o `T:VC1,IR1`

Hasta el 05/10/2026 los nombres que empiezan con `SIRU` empezaban con `SIMUL2` (`SIMUL2_PASO=[0.05]`, `SIMUL2 ADAPTIVE STEP = 0`). Un circuito que todavía los trae así corre igual: `d2datdinc` los lee como los de ahora, y con `-v` lo avisa.

Un `.d` sin `INTEGRATION ROUTINE` usa XPO, como SCEPTRE; en Siru se selecciona con `INTEGRA=[216]`. Un `.ds` sin selector explícito usa Euler (0) y `siru` lo dice.

El paso, si el archivo no lo dice: la milésima parte del tiempo final, y **adaptativo**, que es lo que hace SCEPTRE —el programa para el que se escribieron estos archivos— cuando no le dan controles de paso. Hasta el 06/10/2026 era 0.1 fijo, un número que no tiene nada que ver con la escala de cada circuito: con él los planetas (`STOP TIME = 0.1`) hacían toda la corrida en un paso, `auditorio` (120 horas) pedía 4,3 millones de pasos y los rígidos ($\lambda = -25$, escala 0.04) no convergían. Un paso fijo hay que pedirlo: `SIRU ADAPTIVE STEP = 0` (`.d`) o `SIRU_ADAPT=[0]`

(`.ds`), y conviene dar también el paso, porque si no es esa misma milésima parte. **Cualquier control de paso o de tolerancia prende la adaptación.** Con adaptación y sin `MAXIMUM STEP SIZE`, el paso máximo es el de SCEPTRE: 2 % del tiempo final. Los métodos identificados de `rk.siru` (los índices no son consecutivos):

N	método	N	método
0	Euler	10	Bogacki–Shampine
1	Euler implícito	11	Punto medio implícito
2	Punto medio explícito	12	Lobatto IIIA (trapecio; `TRAP`)
3	Heun	13	Lobatto IIIC, stiff (`IMPLICIT`)
5	Ralston	14	Lobatto IIIC*
6	Kutta	15	Runge–Kutta–Fehlberg
7	Runge–Kutta clásico (`RUK`)	16	Cash–Karp
		17	Dormand–Prince
		18	Gauss–Legendre 6

El índice 4 del catálogo histórico carece de identificación de método (`No se`); 8 y 9 no están definidos. Usar los índices identificados de la tabla. La numeración no expresa el orden del método. Los métodos implícitos 12 y 13 son opciones para circuitos rígidos; el resultado depende también del paso, las tolerancias y el modelo. `INTEGRA=[-1]` pide resolver un punto sin integrar, incluso si el modelo tiene estados: se fijan en sus valores iniciales. Véase §12.

`OUTPUTS` puede pedir cualquier variable del circuito (`VC1`, `IR1`, `QC1`, `IL1`, `VS1`), un estado propio, un parámetro, `T`; lo que no se pueda traducir queda anotado con `-` y no impide simular. Sin `OUTPUTS` salen los estados.

Las semillas. Calcula resuelve cada paso con Newton, y arranca de la semilla de cada incógnita: la solución del paso anterior, o al principio el valor del `.din`, que para las variables de las ramas es 0. Si el modelo no se puede evaluar ahí --una bobina `L2 = 1/VL2` da 1/0 con `VL2 = 0` (`circuito-T3.ds`)--, calcula falla antes de iterar. `SEMILLA VL2 = 0.1` en `RUN CONTROLS` (un `.d` o un `.ds`, una línea por variable) le da otro punto de partida; en un `.ds`, también un valor en `VARIABLES` para la tensión o la corriente de una rama. No cambia el problema, sólo de dónde arranca Newton. Si el nombre no es una incógnita (un parámetro, un estado), se avisa por la salida de errores y se ignora.

`T`, `DT` y `TICK` los pone el simulador en cada paso: el tiempo, el paso en uso y la cantidad de pasos dados. Lo que el archivo les ponga (`CONSTANTS DT=0.01`) no fija nada.

4.12. 11 bis. El punto de operación (`RUN INITIAL CONDITIONS`)

Como en SCEPTRE: con `RUN INITIAL CONDITIONS` en `RUN CONTROLS` (sólo en un `.d`), siru calcula los estados en t = 0 con las derivadas en cero —capacitores abiertos, bobinas en corto— partiendo de las condiciones iniciales del archivo como semilla, y arranca la corrida desde ahí. Con `RUN INITIAL CONDITIONS ONLY` escribe ese punto, una fila, y termina. Es lo que SCEPTRE llama «initial conditions calculation» (manual 2.2.5, 2.2.7), y la base de la sensibilidad, Monte Carlo y el peor caso (pendientes, `PLAN_DC_RERUN.md`). Ejemplo: `siru/corpus/corpus/samples/a03.d`.

Cuando Newton no llega al punto de operación (varios transistores, `exp(38·v)`), o cuando el archivo dice `RUN IC VIA IMPLICIT`, siru lo busca **integrando**, como SCEPTRE: integra el circuito con el tiempo quieto en el inicial, desde las condiciones iniciales del archivo, en tramos del largo de la corrida (hasta 20), y desde el estado de cada tramo prueba Newton sobre el sistema del punto de operación. En la pantalla: `punto DC por integracion: tramo 1 (50 de tiempo), el estado se mueve 3.52 por tramo; Newton llega`. El primer paso de la corrida, el instante de cada fila de salida y el equilibrio arrancan Newton de esa solución y no de las semillas del archivo. Ejemplo: `siru/corpus/corpus/samples/d04.d`.

4.12.1. Estados que quedan como los da el archivo (`IC_DADAS`)

Con `SCEPTRE_IC=[1]` el punto de operación resuelve todos los estados y pisa los valores iniciales que dé el archivo. `IC_DADAS=[1]` cambia eso (18/11/2026): los estados cuyo valor inicial está en `DATOS` (el estado, o la tensión o la corriente de su rama: `VC2 = 2`) quedan en ese valor, y el punto resuelve solo los demás. Es el `.ic` de SPICE, la tensión inicial de un nodo: `cir2dsa` la lleva a los condensadores que tocan ese nodo y pide `IC_DADAS`; y a toda corrida `.tran` sin `uic` le pone `SCEPTRE_IC=[1]`, porque SPICE parte del punto de operación. Prueba: `tests/ic_dadas_e2e.t` en siru (contra ngspice 47).

4.13. 11 ter. Sensibilidad (`RUN SENSITIVITY`)

Como en SCEPTRE (manual 2.2.8), sólo en un `.d`: bajo `CIRCUIT DESCRIPTION` una subsección `SENSITIVITY` con conjuntos `(dependientes / independientes)`, y `RUN SENSITIVITY` en `RUN CONTROLS`:

```
SENSITIVITY
   (VC1, PABC / R1, E1)
RUN CONTROLS
   RUN SENSITIVITY
```

siru calcula el punto de operación DC (11 bis) y, para cada dependiente y cada independiente, la derivada parcial por diferencias centradas y la normalizada `(dy/dx)(x/y)`, y las escribe en `NOMBRE.dc`, una tabla de texto. Dependientes: `VC`, `IR`, `VR`, `IL`, `IE`, una fuente de corriente (su corriente) o un parámetro definido (`PABC`). Independientes: un valor que sea un dato —una resistencia, una fuente, un parámetro definido con un número—; una rama que vale una expresión no lo es, y hay que pedir la derivada respecto de sus parámetros. Ojo: la tabla de sensibilidad de SCEPTRE no coincide con sus propias corridas (`PLAN_DC_RERUN.md`, paso 3); la de siru sí.

4.14. 11 quater. Monte Carlo (`RUN MONTE CARLO`)

Como en SCEPTRE (manual 2.2.9), sólo en un `.d`: una subsección `MONTE CARLO` con conjuntos `(dependientes / independientes)`, cotas en los independientes —`R1 = 100 (90, 110)` mínimo y máximo, o `R1 = 100 (10)` en porcentaje— y `RUN MONTE CARLO = n` en `RUN CONTROLS` (10 si no se da `n`):

```
MONTE CARLO
   (VC1, IR2 / R1, E1)
RUN CONTROLS
   RUN MONTE CARLO = 50
   INITIAL RANDOM NUMBER = 127263527
   DISTRIBUTION = UNIFORM
```

siru hace exactamente lo que SCEPTRE: `n` puntos de operación DC (11 bis) con cada independiente en `media + sigma·g`, media `(max+min)/2`, sigma `(max−min)/6`, `g` gaussiana (suma de 12 uniformes menos 6) o uniforme; el generador es el de SCEPTRE (RANDU) con su semilla,

así que con la misma semilla salen las mismas muestras. Escribe en `NOMBRE.mc` una tabla con, por independiente y dependiente, el nominal, el mínimo, el máximo, la media y el sigma muestral (con `N−1`). Ojo, dos cosas heredadas de SCEPTRE: la "uniforme" cae en `[media, media+sigma)`, no entre las cotas; y tras `RUN MONTE CARLO` los OUTPUTS que lista SCEPTRE son los de la última muestra, los de siru los nominales. `siru/corpus/sceptre` compara `NOMBRE.mc` con las OBSERVED STATISTICS de SCEPTRE (`NOMBRE.sceptre-mc`); en a07 coinciden en ocho cifras.

4.15. 11 quinquies. Peor caso (`RUN WORST CASE`)

Como en SCEPTRE (manual 2.2.7), sólo en un `.d`: una subsección `WORST CASE` con conjuntos `(dependientes / independientes)`, cotas en los independientes (como en Monte Carlo) y `RUN WORST CASE` en `RUN CONTROLS`:

```
WORST CASE
   (VCET1, VCCT1 / P1T1, P1T2, R3)
RUN CONTROLS
   RUN WORST CASE
```

Por cada dependiente, siru calcula el gradiente respecto de los independientes en el punto de operación nominal, lo normaliza y avanza por esa recta hasta la primera cota que toca, a favor del gradiente (HIGH) y en contra (LOW); en esos dos puntos resuelve el punto de operación. Es exactamente el método de SCEPTRE (`worst.f`): no pone cada independiente en su cota, sólo el más cercano llega a la suya. Escribe `NOMBRE.wc`: por (dependiente, independiente) el nominal, HIGH y LOW del dependiente, y el valor y el gradiente del independiente en los tres puntos; en comentarios, la distancia y la cota que toca cada extremo. Ojo: los extremos de siru no son los de SCEPTRE, porque el gradiente de SCEPTRE está mal (11 ter); en a09 los de siru son más extremos, y SCEPTRE corrido en los puntos de siru da lo mismo que siru (`PLAN_DC_RERUN.md`, paso 5). Con `RUN WORST CASE = LOW | HIGH` la corrida en el tiempo sigue con los valores de ese extremo del último dependiente (y desde su punto de operación); con `NOMINAL`, por omisión, con los nominales. Ojo: SCEPTRE integra con los valores contrarios (con HIGH deja los LOW y al revés; `worst.f` calcula HIGH y después LOW y queda lo último): siru hace lo que dice el manual.

4.16. 11 sexies. Reruns (`RERUN DESCRIPTION`)

Como en SCEPTRE (manual 2.4), sólo en un `.d`: después de `RUN CONTROLS`, `RERUN DESCRIPTION (N)` con las secciones que cambian y, en cada línea, los N valores, uno por rerun:

```
RERUN DESCRIPTION (2)
ELEMENTS
   R1 = 9.3, 9.5
DEFINED PARAMETERS
   PF = 500, 600
INITIAL CONDITIONS
   VC1 = 1, 2
FUNCTIONS
   TABLE 1 = 0, 0, 0
             40, 1.2, .4
RUN CONTROLS
   STOP TIME = 600, 700
   INTEGRATION ROUTINE = XPO, RUK
   IC FOR RERUNS = PRECEDING RESULTS
```

siru corre la maestra y después cada rerun con sus cambios (elementos, parámetros, condiciones iniciales, tablas, `STOP TIME`, `START TIME`, `INTEGRATION ROUTINE`; lo demás se avisa y no

se aplica). Cada rerun parte de las condiciones iniciales del archivo (`IC FOR RERUNS = MASTER RESULTS`, por omisión) o del estado final de la corrida anterior (`PRECEDING RESULTS`). Todo va al mismo `NOMBRE.simul`, un bloque por corrida con su cabecera y una línea `#  RERUN k/N (R1 = 9.3, STOP TIME = 600)` (0 es la maestra). Para graficar una corrida con el modelo v2 debe seleccionarse explícitamente su bloque en el DAT; Tabular2 lo exporta sin mezclar las corridas. Cada bloque comienza en `#  Archivo.` `siru/corpus/sceptre` compara bloque por bloque; a03 y tanque_par_1 coinciden con SCEPTRE (`PLAN_DC_RERUN.md`, paso 6).

4.17. 11 septies. Seguir una corrida (`CONTINUE`, `--continuar`)

Una corrida terminada se sigue desde donde quedó:

```
siru --continuar --hasta 40 circuito.d
```

siru lee el estado final que dejó en `circuito.informe` (`estado final: t=20 qc1=6.87`; si no está, la corrida no terminó bien y hay que correrla de nuevo), integra de ahí hasta 40 y anexa el bloque `#  CONTINUE (desde t = 20, STOP TIME = 40)` al mismo `circuito.simul`; el `.informe` queda con el nuevo estado final, así que se puede volver a continuar. No se recalcula el punto de operación ni las opciones DC; los valores que dejó el peor caso (`WORST CASE = LOW | HIGH`) se heredan. En un `.d`, la sección `CONTINUE` de SCEPTRE (manual 2.5) con sus `RUN CONTROLS` (`STOP TIME = 2000`) hace lo mismo al final de la corrida, siguiendo la última (la maestra o el último rerun):

```
CONTINUE
RUN CONTROLS
   STOP TIME = 2000
```

4.18. 11 octies. El análisis en frecuencia (`RUN AC`)

`RUN AC` (manual de SCEPTRE 2.2.4) no integra en el tiempo: linealiza el circuito alrededor de su punto de operación y resuelve en frecuencia. Es un análisis de pequeña señal.

```
ELEMENTS
  EDC, GND-1 = 0.6                 la polarización, como siempre
  EAC, 1-2 = (0.001, 0.)           la excitación: magnitud y fase en grados
  ...
RUN CONTROLS
  RUN AC
  INITIAL FREQUENCY = 100
  FINAL FREQUENCY = 10000
  NUMBER FREQUENCY STEPS = 2       dos intervalos, o sea tres frecuencias
  TYPE FREQUENCY RUN = LOG         o LINEAR (el que se toma si no se dice)
```

Una fuente que vale `(magnitud, fase)` es la excitación. En el tiempo vale 0: es una perturbación, no una fuente de la corrida. Pueden ser varias, y sus respuestas se suman.

El punto de operación se calcula siempre, aunque el archivo no diga `RUN INITIAL CONDITIONS`: el análisis es alrededor de él. Si el archivo no trae `STOP TIME`, el análisis es toda la corrida y no hay nada en el tiempo, como con `RUN INITIAL CONDITIONS ONLY`.

La salida es `NOMBRE.ac`: una fila por frecuencia y, por cada variable de `OUTPUTS`, cuatro columnas, su parte real, su imaginaria, su módulo y su fase en grados. El encabezado dice la excitación y los estados.

Cómo lo hace (`Siru::AC`): el circuito es `dq/dt = f(q, u)` con las salidas `y = g(q, u)`, y hace falta `A = df/dq`, `B = df/du`, `C = dg/dq` y `D = dg/du`. Esas derivadas salen por diferencias centradas del sistema del punto de operación, el que tiene las derivadas como datos y los estados como incógnitas: una corrida por cada dato que se mueve. No se deriva ninguna expresión.

Después, para cada frecuencia, `(sI − A) X = B U` y `Y = C X + D U`, con `s = j·2πf` y `U` el fasor; el sistema complejo se resuelve por eliminación gaussiana con pivoteo.

No se usa el sistema del instante, el que tiene los estados como datos, porque en un circuito con transistores no converge: dada la carga de una juntura, hallar su tensión es el problema difícil.

Probado (`tests/ac.t`, los circuitos en `corpus/corpus/inventados/ac-*.d`): un pasabajos RC coincide con su transferencia `1/(1 + jωRC)` en $6 \cdot 10^{-11}$, y un diodo polarizado, que no tiene fórmula, coincide con la corrida en el tiempo del mismo circuito excitado con `0.001·sin(2πft)` en $2{,}5 \cdot 10^{-5}$, que es el error del paso.

Lo que no hace:

`PRINT EIGENVALUES`, `PRINT EIGENVECTORS`	los valores propios del circuito linealizado; se avisa
`COMPLEX`, `NYQUIST`, `DEGREES`, `RADIANS` en `OUTPUTS`	el formato de columnas de SCEPTRE; el `.ac` trae siempre las cuatro
`(10,4)`, `COMPLEX` en el valor	el fasor dado por parte real e imaginaria en lugar de magnitud y fase
parámetros y funciones complejas	lo que prueba `complex.d`: aritmética compleja en las expresiones

4.19. 11 nonies. Medidas sobre la corrida (`MEDIDAS`)

El bloque `MEDIDAS`, en el archivo de datos y usos, pide números que resumen la corrida: un máximo, un promedio, el instante de un cruce, una demora (18/11/2026; son las de `.measure` de SPICE). Siru las calcula al terminar, sobre la tabla de resultados, y las deja en la pantalla y en el `.informe`, un renglón `medida NOMBRE = valor` por cada una. Lo que se mide tiene que estar en `RESULTADOS`.

```
MEDIDAS
  vmax   = {"tipo": "maximo", "de": "U2", "desde": 0.002, "hasta": 0.003}
  t90    = {"tipo": "cruce", "de": "U2", "igual": 0.9, "sentido": "sube", "numero": 1}
  demora = {"tipo": "demora", "inicio": {"de": "U1", "igual": 0.5, "sentido": "sube"},
                              "fin":    {"de": "U2", "igual": 0.5, "sentido": "sube"}}
  doble  = {"tipo": "cuenta", "expresion": "2*vmax"}
FIN
```

`tipo`	Qué mide	En SPICE
`maximo`, `minimo`, `pico_a_pico`	De la columna `de`, entre `desde` y `hasta` (si faltan, toda la corrida).	`MAX`, `MIN`, `PP`
`t_maximo`, `t_minimo`	El instante donde está el máximo o el mínimo.	`MAX_AT`, `MIN_AT`
`promedio`, `rms`, `integral`	Por trapecios, entre `desde` y `hasta`.	`AVG`, `RMS`, `INTEG`
`valor`	`de` en el instante `en`, o en el cruce `cuando`.	`FIND ... AT`, `FIND ... WHEN`
`derivada`	La pendiente de `de` en el instante `en`.	`DERIV`
`cruce`	El instante en que `de` cruza `igual` (un número u otra columna). `sentido`: `sube`, `baja` o `cualquiera`; `numero`: 1, 2… o `"ultimo"`; `desde` y `hasta` acotan dónde se busca.	`WHEN`

tipo	Qué mide	En SPICE
demora	El tiempo del cruce inicio al cruce fin; cada uno es un cruce o {"en": t}.	TRIG ... TARG
cuenta	Una cuenta con otras medidas, por su nombre (y con columnas, que valen lo de la última fila).	param=, let
falla	Una medida que falla siempre, con su motivo: la que SPICE rechaza por mal escrita.	—

Las cuentas admiten números, nombres, + - * / ^, paréntesis, comparaciones (< <= > >= == !=), &&, ||, !, condición ? si : no y las funciones abs, sqrt, exp, ln, log10, sin, cos, tan, atan, floor, ceil, nint (el entero más cercano), sgn, min, max y pow. de puede ser una cuenta entre columnas ("U2*U1"), que se hace fila por fila. igual, desde, hasta y en pueden ser un número o una cuenta con otras medidas ("0.9*ymax"); igual también otra columna o una cuenta entre columnas. Una medida puede nombrar a otra en cualquier orden; si se nombran en ronda, corta. Las cuentas y las verificaciones también pueden nombrar los parámetros del archivo que son un número y, si la corrida hizo un análisis en frecuencia, sus columnas de la última frecuencia: U3_re, U3_im, U3_mag, U3_fase y U3_ac, el número complejo.

Entre dos filas se interpola en línea recta, como SPICE. Una columna se puede nombrar con un menos adelante ("-IV1") y se mide cambiada de signo. Una medida que no se puede calcular (no hay cruce, la ventana queda afuera) no tiene valor y dice por qué, sin cortar la corrida; una mal escrita, o de una columna que no está en los resultados, corta. cir2dsa escribe este bloque de las .measure tran de un circuito de SPICE, y de los let de su guion .control. Pruebas: tests/medidas.t y tests/medidas_e2e.t en siru (contra ngspice 47).

4.19.1. Verificaciones (VERIFICACIONES)

El bloque VERIFICACIONES, también en el archivo de datos y usos, dice lo que la corrida tiene que cumplir (18/11/2026): condiciones con las medidas, por su nombre, y con las columnas de la tabla, que valen lo de la última fila.

```
VERIFICACIONES
  carga = {"condicion": "abs(vmax - 5) < 0.01", "mensaje": "el condensador no llega a
    5 V"}
  orden = {"condicion": "t90 > tsube"}
FIN
```

Siru las revisa al terminar, después de las medidas, y deja un renglón por cada una en la pantalla y en el .informe: verifica carga: bien o verifica orden: NO SE CUMPLE t90 > tsube. Una verificación se cumple si su condición da distinto de cero; si nombra una medida sin valor, dice sin valor y por qué. Que una no se cumpla no cambia el estado de salida de siru. cir2dsa escribe este bloque de los if ... echo "ERROR ..." ... end del guion .control de un circuito de ngspice: la verificación es que no se dé la condición que lleva al error.

4.20. 11 decies. El espectro de una curva (ESPECTROS)

El bloque ESPECTROS, en el archivo de datos y usos, pide la transformada de Fourier de una curva de la corrida (18/11/2026; es el linearize y el fft de SPICE, con la misma cuenta que ngspice 47).

```
ESPECTROS
  e1 = {"de": "U1", "paso": 2e-6}
  e2 = {"de": "U3", "paso": 2e-6, "ventana": "hanning", "desde": 0, "hasta": 0.05}
FIN
```

`de` es la columna de la tabla de resultados (con un menos adelante, cambiada de signo); `paso`, el de la grilla pareja a la que se lleva la curva interpolando en línea recta; `desde` y `hasta`, el tramo (si faltan, toda la corrida); `ventana`: `hanning` (la de omisión), `hamming`, `rectangular`, `none`, `triangle`, `blackman` o `flattop`; `tipo`: `fft` o `psd` (la densidad espectral de potencia del `psd` de ngspice, en V^2/Hz, suavizada con `suavizado` puntos, a `NOMBRE.psd`). Siru lo calcula al terminar y lo deja en `NOMBRE.fft`, con la forma del `.ac`: una fila por frecuencia y por cada espectro su parte real, la imaginaria, el módulo y la fase en grados. Con `L` puntos, la frecuencia `k` es `k / (L paso)` y el valor es la transformada dividida por `L/2` (la de frecuencia cero, por `L`): una senoidal de amplitud 1 da 1 en su frecuencia. Todos los espectros de un archivo tienen que tener la misma grilla.

4.21. 11 undecies. Curvas hechas de otras (`CURVAS`)

El bloque `CURVAS`, en el archivo de datos y usos, pide curvas que salen de otras después de la corrida (18/11/2026): `NOMBRE = {"tipo": ..., "de": columna}`, con `tipo` `promedio_movil` (y su `ventana`: el promedio en $(t - \text{ventana}, t]$, el `mtimeavg` de SPICE), `derivada` (`deriv`) o `integral` (`integ`). Siru las deja en `NOMBRE.curvas`, una fila por fila de la tabla.

4.22. 11 duodecies. El guion de vectores (`GUION`)

El bloque `GUION`, en el archivo de datos y usos, es un programa que siru corre **después de la simulación** sobre la tabla de resultados (18/11/2026): cada columna es un **vector** —la curva entera— con el que se hacen cuentas, se leen y escriben elementos, se recorre con bucles y se transforma ida y vuelta. Es álgebra sobre los resultados, para lo que no se reduce a una medida (§4.19), una curva (§4.21) o un espectro (§4.20), y sirve para cualquier modelo. Un renglón del programa por entrada, entre comillas, en orden de nombre:

```
GUION
  p0001 = "let dt = tiempo[1] - tiempo[0]"
  p0002 = "let horas = sum(TLOCAL > 26) * dt / 3600"
  p0003 = "let f = fft(TLOCAL + j(0))"
  p0004 = "let f[0] = 0"
  p0005 = "let sin_media = real(ifft(f))"
FIN
```

Renglones. `let X = cuenta`; `let X[k] = cuenta` (un elemento, contando desde 0); `unlet X`; `compose X start=a stop=b step=c` (o `lin=n`, o `values a b c`); `linearize PASO DESDE HASTA` (todas las columnas a paso fijo); `while C`, `dowhile C`, `repeat N`, `if C`, `else`, `end`, `break`, `continue`; `echo texto` (`$&X` es el valor de `X`); `print cuenta`.

Cuentas. Un nombre que el guion no definió es una columna de `RESULTADOS`; `tiempo` es la primera. Elemento a elemento, y un número vale para todos. Operadores `? :`, `|`, `&`, `!`, las comparaciones (que dan 1 o 0 por elemento), `+ - * / % ^` y el índice `[k]`. Funciones: `abs`, `mag`, `real`, `imag`, `ph`, `db`, `sqrt`, `exp`, `ln`, `log` (el natural, como `ln`), `log10`, `sin`, `cos`, `tan`, `atan`, `j`, `mean`, `sum`, `vecmax`, `vecmin`, `norm`, `length`, `unitvec`, `vector`, `pos`, `fft`, `ifft`. Los vectores pueden ser complejos. Una condición vale si algún elemento no es cero.

Resultados. Cada número queda en el informe (`guion NOMBRE = valor`), cada `echo` también, y los vectores en `NOMBRE.vectores`, una fila por índice. Un error del guion corta la corrida diciendo el renglón. El detalle, con ejemplos y usos, está en el manual de siru (*Álgebra de resultados*). El nombre del bloque y la forma de los renglones son los de esta primera versión.

4.23. 12. Un punto, sin integración: INTEGRA=[-1]

El `.ds` contiene la información necesaria para Calcula: incógnitas, semillas, datos conocidos, constantes, funciones y ecuaciones; si hay circuito, también sus ramas y modelos. Con

`INTEGRA=[-1]`, **Siru construye el sistema, llama a Calcula para resolverlo una vez y pone sus resultados en `.simul`**. No hace falta escribir un `.din` aparte, preparar una simulación temporal ni declarar un tiempo ficticio.

Aspecto	Comportamiento con `INTEGRA=[-1]`
Lugar y escritura	En `RUN_CONTROLS` o `RUN CONTROLS`, una línea `INTEGRA=[-1]`. Es sintaxis `.ds`; no escribir `KR`, `rk=-1` ni quitar los corchetes.
Resolución	Una evaluación del sistema de Calcula. Calcula puede hacer las iteraciones internas necesarias para converger.
Incógnitas	Los valores de `VARIABLES` son semillas. Los valores finales salen de resolver las ecuaciones.
Datos	Los parámetros numéricos quedan fijados en los valores dados.
Estados	Si existen, se fijan en sus valores iniciales; se pueden calcular sus derivadas, pero no se avanza en el tiempo.
Controles temporales	No se hace un barrido aunque permanezcan `T`, `TIEMPOFINAL`, un paso o condiciones de continuación en el archivo.
Otros análisis	Este modo no lanza los barridos/análisis DC adicionales. Para esos análisis usar el modo y los controles de §11 bis–quinquies.
Salida	Una fila numérica con todas las variables declaradas en el sistema enviado a Calcula, incluidos sus datos. `OUTPUTS` no recorta esta tabla de un punto.
Precisión	Los números se copian tal como los devuelve el protocolo de Calcula, sin un segundo redondeo ni sustituir resultados ausentes por cero. Algunos números pueden mostrar muchos decimales: ésa puede ser la escritura que entrega Calcula.
Nombres	Se usan los nombres del sistema generado, habitualmente en minúsculas. Si `OUTPUTS` conserva nombres originales distintos, se agrega la fila `sceptrevars` para que Simplot pueda relacionarlos con la fuente. No se duplica una fila de nombres idénticos.
Unidades	Se escribe la fila de unidades sólo cuando el sistema declara alguna. Para columnas sin unidad, dentro de esa fila, se usa `-`; si ninguna tiene unidad, se omite toda la fila. No se inventan `??`, magnitudes ni unidades a partir del nombre de una variable.
Dibujo	Si hay ramas y una sección `FORMAS`, incluso vacía, también se solicita el SVG al paquete opcional `dibuja`.

Aspecto	Comportamiento con `INTEGRA=[-1]`
Limitación	El modo de un punto con llaves `SB` se rechaza: requiere un tratamiento de las conmutaciones que este modo no implementa.

Limitación actual: las cantidades que el lector convierte a SI no conservan necesariamente una etiqueta de unidad en cada variable del sistema generado. La migración de §3.9 debe conservar esa información e interpretar `UNIDADES` y `RESULTADOS` en Slex al construir el `.dat`, antes de simular. Los pedidos de salida se aplican cuando existen los resultados; no son secciones exclusivas de Simplot.

Ejemplo completo, autosuficiente, sin bibliotecas externas, circuito ni tiempo. Guardarlo como `punto.ds`:

```
CIRCUIT_DESCRIPTION
Un dato y una incognita: X al cuadrado es A.
DEFINED_PARAMETERS
A=4
VARIABLES
X=1
EQUATIONS
X^2-A
OUTPUTS
X,A
RUN_CONTROLS
INTEGRA=[-1]
END
```

Al ejecutar `siru punto.ds`, Calcula resuelve `X^2=A` desde la semilla `X=1` y entrega la raíz positiva, aproximadamente 2. `punto.simul` contiene los nombres `a`, `x`, sus nombres originales `A`, `X` y una fila numérica con 4 y el resultado de Calcula. No hay columna de tiempo ni fila de unidades porque no se declararon. Si ya existe la salida, Siru conserva la anterior y usa `punto-1.simul`, luego `punto-2.simul`, etc.

La llamada automática a Simplot descrita en §2 puede presentar la tabla al terminar. Para ejecutar exclusivamente el cálculo y escribir la tabla, usar `siru -nosimplot punto.ds`; esto no cambia el significado de `INTEGRA=[-1]` ni sus resultados.

Un ejemplo real es psicro_r_srk.M.dsa (`../../../corpus/corpus/lolipmp/psicro_r_srk.M.dsa`, con su `.DU.dsa` al lado; era `psicro_r_srk.ds`), copia de `psicro_5.ds` con `INTEGRA=[-1]` explícito. `TEMP`, `TBH` y `PRES` son datos; `XXX`, `WWW` y `PV` son incógnitas. Sus ecuaciones y funciones están en el mismo archivo. Se ejecuta desde la raíz del repositorio con:

```
siru siru/corpus/corpus/lolipmp/psicro_r_srk.ds
```

Sin `INTEGRA=[-1]`: un circuito sin estados y sin corrida temporal también se resuelve automáticamente en un punto, como `slex/d2datdinc/pruebas/nulo.d`. Si no tiene estados pero sí pide una corrida temporal, Siru puede resolver un punto por cada `T` de la grilla, usando la solución anterior como semilla del siguiente. Eso es un barrido y produce varias filas; no es el modo explícito `INTEGRA=[-1]`.

4.24. 13. Los mensajes, y qué hacer

Dónde aparece un error. En tres lugares, del más corto al más completo:

1. En la pantalla, la última línea: `siru: NO CONVERGE en t=78.3 (paso 783)`, o `siru: no arranca en t=0 (paso 0)`.

2. En `NOMBRE.informe`: `resultado:`, `hasta:` (el paso y el tiempo, contra el `TIEMPOFINAL`) y `error:`, la primera línea del motivo.
3. En `NOMBRE.log`: todo, sin cortar. Cada línea dice quién la escribió: `d2datdinc: ...` (la traducción), `calcula: ...` (el sistema y el solver, y adentro de él tabular2 y las funciones externas) y al final `siru: resultado: ....`

`siru/corpus/regenera` deja el mismo `NOMBRE.log` al lado de cada fuente, con lo que dicen `d2datdinc` y `clasificar` al regenerarla, y `siru/corpus/clasificar` copia el resultado de `NOMBRE.informe` al `.tipo` (campo `siru`).

Los resultados del `.informe`:

resultado	qué quiere decir
`llega al final`	integró hasta `TIEMPOFINAL` (o resolvió el punto)
`NO CONVERGE`	arrancó y en algún paso posterior al primero el solver no encontró solución; el `hasta:` dice dónde. Las filas hasta ahí son válidas
`no arranca`	falló antes de integrar o en el paso 0: la traducción, la carga del sistema, o el solver en el punto inicial

Las tablas. Un error de `tabla()` lo detecta tabular2 y lo informa `calcula`, con el código, su motivo y el dato concreto:

```
calcula: tabla('jx','X','Y'): error -3 de tabular2: columnas mal especificadas:
la columna de abscisas (la X) tiene que crecer de una fila a la siguiente
(puede repetirse, no bajar), y las columnas pedidas tienen que existir
(la columna 0 de la tabla (la X) baja en la linea de datos 3: 1 despues de 2)
```

Los códigos de tabular2 (`tabular2/src/table.h`; el motivo lo da `tb_mensaje()`, el dato `tb_detalle()`, desde tabular2 2.0-10):

código	motivo	qué hacer
-1	no se pudo leer la tabla: no tiene el formato de tabular2	una línea `#` con los nombres de columna y después los datos
-2	no se pudo abrir el archivo	ver la ruta: una relativa se busca junto al archivo que la usa
-3	la X no crece de una fila a la siguiente, o las columnas pedidas no existen	ordenar la tabla; el detalle dice en qué línea baja
-4	la misma columna como X y como Y	pedir dos columnas distintas
-5	datos no válidos (algo que no es número, un número de tabla repetido en `etable.txt`)	corregir el dato
-6	argumento no válido (método distinto de 1 o 2, un número negativo)	método 1 lineal o 2 spline

código	motivo	qué hacer
−7	la tabla no tiene datos en esas columnas	revisar la tabla
−8	error de entrada o salida	el disco, los permisos
−15	sin memoria	
−888	no hay una columna con ese nombre en el encabezado	usar el nombre de la línea `#`
−889	no se pudieron relacionar esas dos columnas	la X no crece, o faltan datos

Las funciones externas (`EXTERNAL NOMBRE (N) RUTA`, sección 9):

mensaje	qué pasa	qué hacer
`el .din declara funciones externas (codigo de afuera); calcula las carga solo con CALCULA_EXTERNAS=1`	calcula no carga código de afuera sin permiso	`CALCULA_EXTERNAS=1` en el entorno
`gfortran no pudo compilar RUTA`	el fuente tiene un error, o no hay `gfortran`	compilarlo a mano: `gfortran -shared -fPIC RUTA`
`la biblioteca de RUTA no tiene la funcion nombre_`	el nombre del `EXTERNAL` no es el de la `FUNCTION`	que coincidan
`NOMBRE pide N argumentos y le dieron M`	la cantidad del `EXTERNAL` no es la del llamado	corregir la `N`

Los demás:

mensaje	qué pasa	qué hacer
`PENDIENTE ...`	algo del archivo no se pudo traducir; no se simula	mirar qué es: un símbolo que no está, un control que Simul no tiene
`tipo X de R: no tiene modelo en el dato`	un `.ds` sin `include` de la biblioteca, o un tipo que no existe	agregar el `include` o declarar el tipo
`modelo por incrementos`	`L` o `C` de la biblioteca vieja	usar `LB`, `CB`
`valor de R: no se puede traducir el simbolo X`	`X` no es rama, parámetro, función ni `T`	declararlo
`la funcion F no esta en el dato`	se usa sin definir en `FUNCTIONS`	definirla
`grados de libertad ...`	las ecuaciones propias no cierran con sus incógnitas	una `VARIABLE` por ecuación, o una ecuación menos
`aviso: los nodos 1 2 no tienen camino a la masa (GND)`	una parte del circuito no toca la masa (una isla)	no es un error: en la isla un nodo va a potencial cero, de referencia; las tensiones y corrientes de las ramas no cambian. Si es un descuido, conectarla

mensaje	qué pasa	qué hacer
`los datos dados no determinan el sistema: ... incognitas que ninguna ecuacion libre siente`	Calcula: hay una incógnita que ninguna ecuación fija (un nodo que flota, una ecuación repetida)	mirar las que nombra
`no se obtuvo una solucion valida ... not making progress`	Newton no converge desde la semilla (diodos con mucha ganancia)	semillas razonables en `DEFINED PARAMETERS`; método 13
`la capacidad de C depende de X: en un lazo de capacitores`	en un lazo sólo puede depender del voltaje de capacitores (el propio u otro) o de una fuente en paralelo con uno (sección 7)	sacar el lazo o la dependencia
`vale (1,0), un fasor: es el analisis AC de SCEPTRE`	Diagnóstico de versiones antiguas sin AC	Actualizar y usar el análisis AC de §11 octies; respetar sus límites.
`se usa el metodo 0` para en `TICK<=1000`	un `.ds` sin selector explícito la parada que puso el archivo	`INTEGRA=[112]` es lo pedido

4.25. 14. Lo que no se puede

- `OPTIMIZATION` no se implementa. `RUN AC` sí está implementado con las restricciones de §11 octies; no implica admitir aritmética compleja general en las expresiones.
- Los controles de error de SCEPTRE (`MINIMUM/MAXIMUM ABSOLUTE/RELATIVE ERROR`): no son las tolerancias de siru.
- Los modelos por incrementos de la biblioteca vieja (`L`, `C`, `R`, `E` sin `B`).
- Tablas con `x` no creciente.
- Expresiones en FORTRAN o con multiplicación sobreentendida: van en ExprTk.
- Un salto de corriente en una bobina acoplada por `M` no se transmite a la otra si no está en un corte.
- Varias ramas con el mismo nombre.

4.26. 15. PARAPLOT completo: fuentes, series, paneles y unidades

4.26.1. 15.0 Grupos PARAPLOT y resultado GNUPLOT

Un `.ds` puede contener varias secciones `PARAPLOT [NOMBRE]`. El nombre es opcional; se conserva exactamente, incluidas mayúsculas y minúsculas. Si falta, se asigna el número de posición de esa sección entre todos los PARAPLOT, contados desde 1. Los nombres admiten letras, cifras, guion, punto y guion bajo; no espacios. Los identificadores repetidos —también la colisión entre un nombre numérico explícito y uno automático— son errores. Las órdenes de un grupo no se mezclan con las de otro; se pueden repetir los nombres de gráficos interiores en grupos distintos.

Pedido	Resultado de Simplot
Primer grupo: `PARAPLOT TEMPERATURAS`	`GNUPLOT TEMPERATURAS`
Segundo grupo: `PARAPLOT`	`GNUPLOT 2`
Tercer grupo: `PARAPLOT ESTADOS`	`GNUPLOT ESTADOS`
Ningún PARAPLOT	Un único `GNUPLOT 1` automático, usando las tablas.

Simplot tiene dos misiones: **generar** esos GNUPLOT y **ejecutarlos** invocando Gnuplot. Por defecto hace ambas. `--no-generar` conserva y ejecuta los existentes; si no hay ninguno, informa que no puede dibujar. `--no-ejecutar` genera sin dibujar. Ambas opciones conservan el documento sin hacer ninguna etapa. `-noX` solo suprime ventanas; `-save` guarda el documento y los SVG efectivamente producidos. Sin `-save`, el documento sale por stdout. Al regenerar se reemplazan los GNUPLOT anteriores.

Las secciones `GNUPLOT NOMBRE` van al final del documento, una tras otra. Cada cuerpo es texto Gnuplot literal, incluidos comentarios y bloques de datos; termina en el próximo encabezado GNUPLOT o en EOF. Su última línea es una receta de terminal Bash para invocar Gnuplot con **ese grupo** del documento (`gnuplot -persist <(awk ...)`). No lleva `END` ni `END GNUPLOT`. El nombre de archivo utilizado por la receta se elige con `--salida`; si se redirige stdout debe conservarse ese nombre. Slex almacena el texto, no lo ejecuta. Simplot separa la receta y entrega el programa directamente al ejecutable Gnuplot, sin pasar la receta a un shell. Ejecutar programas GNUPLOT recibidos exige confiar en ellos.

El DAT conserva los grupos en `GLOBAL.simplots` (`nombre`, `pedido`, `modelo`) y los programas recibidos en `GLOBAL.gnuplots` (`nombre`, `texto`). `GLOBAL.simplot` conserva el primer modelo para los consumidores anteriores. Las tablas son comunes al documento. Véanse `simplot/corpus/entradas-ds/grupos.ds`, `solo-gnuplot.ds` y `tests/documento.t`. La generación documental todavía no traduce todas las órdenes gráficas históricas; ese recorrido continúa disponible con las opciones de compatibilidad.

4.26.2. 15.1 Qué se dibuja y qué archivo contiene cada cosa

`FORMAS` describe el esquema de ramas. `PARAPLOT` describe cómo representar resultados numéricos: son dos secciones independientes. El `.ds` conserva las instrucciones gráficas, pero una tabla externa sigue siendo un archivo externo; producir el `.dat` no copia sus números dentro de él.

Sección o control	Efecto
`OUTPUTS`	Solicita observables del cálculo. Si no hay descripción gráfica explícita, `d2datdinc --graficos` puede preparar gráficos a partir de esa lista.
`RESULTADOS`	Pedido común de variables, nombres, magnitudes y unidades de salida, interpretado en Slex antes de simular (§3.9). Sin PARAPLOT, sus datos preparados pueden dibujarse automáticamente.
`UNIDADES`	Declaraciones comunes de unidad de origen y magnitud; se resuelven en Slex antes de Siru, no en el graficador (§3.9).
Sin `PARAPLOT`	Simplot dibuja con las columnas y metadatos disponibles; no exige una descripción gráfica explícita.
`PARAPLOT`, `GRAFICOS`, `ACEPTAR_GRAFICOS`	Órdenes gráficas explícitas; normalmente reemplazan los gráficos automáticos de `RESULTADOS`.
`GRAFICOS_ADICIONALES`	Agrega órdenes al final.
`ANADIR_GRAFICOS=[1]` en `RUN_CONTROLS`	Conserva los gráficos automáticos y agrega los explícitos. Se aceptan también `AÑADIR_GRAFICOS` y `AGREGAR_GRAFICOS`, con 0/1 o [0]/[1].

Sección o control	Efecto
`SIMPLOT_VERSION=[1]`	Motor histórico, valor predeterminado.
`SIMPLOT_VERSION=[2]`	Modelo gráfico estructurado; sólo admite las órdenes `.ds` enumeradas en §15.3. También se admite el valor sin corchetes.

Una orden habitual es `comando OBJETO = contenido`. Se escribe una por sentencia y se aplican los comentarios, inclusiones y continuaciones de `.ds`. La excepción `nomostrar G,H` no lleva `=`. Un archivo `.paraplot` independiente entra directamente en esta sección, sin necesidad de escribir su encabezado.

`archivo F = datos.simul` asigna el nombre simbólico `F` a una tabla real. Las rutas relativas se resuelven respecto del documento. Las series se escriben `F,X:Y,Z`: las columnas Y y Z contra X. Los índices comienzan en 1; también pueden usarse nombres de columnas. La abscisa no tiene que ser tiempo. Se pueden superponer datos calculados y medidos declarando varias fuentes.

Siru entrega a Simplot la tabla recién producida y la descripción gráfica, incluso si cambió el nombre de salida. Las fuentes externas conservan sus propias rutas. Si se ejecuta Simplot por separado, la tabla debe estar disponible; un `.ds` sólo con órdenes gráficas no calcula los números.

4.26.3. 15.2 Todas las órdenes del motor histórico (v1)

Orden	Escritura	Significado
`archivo`	`archivo F = ensayo.simul`	Nombre real de la fuente.
`sceptrevars`	`sceptrevars F = T,TEMP,FLUJO`	Nombres de las columnas de origen, según sus metadatos.
`variables`	`variables F = Tiempo,Interior,Flujo`	Leyendas por columna; `_` conserva el nombre de origen.
`magnitudes`	`magnitudes F = tiempo,temperatura,potencia`	Magnitudes asociadas; `_` deja sin nombre.
`unidades`	`unidades F = s,tempC,W`	Unidades de las columnas; `_` para una columna sin unidad. No confundir esta orden con el encabezado `UNIDADES`.
`indices`	`indices F = 1,3,2`	Correspondencia de la lista de variables/metadatos con las columnas físicas.
`unipri`	`unipri F = horas`	Presentación temporal de la primera columna; requiere una unidad de origen temporal. Acepta horas, días, minutos o segundos según el vocabulario histórico.
`arch_var`	`arch_var F = base.simul,caso1.simul,caso2.simul`	Fuentes correspondientes a 2 simulaciones de un mismo problema; la primera coincide con `archivo`.
`variaciones`	`variaciones F = Base,R=2,R=3`	Leyendas que distinguen esas variaciones, una por archivo.

Orden	Escritura	Significado
`selec`	`selec RECORTE = F,1,0,10`	Nueva fuente con filas cuya columna 1 está entre 0 y 10; no modifica el original. Admite encadenar selecciones.
`lineas`	`lineas G = F,1:2,3`	Series unidas por líneas; repetir sobre G agrega series.
`puntos`	`puntos G = F,1:2`	Serie de puntos, apropiada también para una única fila.
`funcion`	`funcion G = sin(x)`	Función de Gnuplot; no es una función de `FUNCTIONS` ni una ecuación de Calcula.
`titulo`	`titulo G = Evolución de la temperatura`	Título del gráfico.
`ejes`	`ejes G = Tiempo(s),Temperatura(tempC)`	Etiquetas y unidades de los ejes; prevalecen sobre las magnitudes generales.
`rangox`, `rangoy`	`rangox G = 0:10`	Rango visible del eje; v1 usa la forma histórica con dos puntos. En v2 se escribe con coma.
`mostrarjuntos`	`mostrarjuntos PANEL = G,H`	Reúne varios gráficos en un cuadro.
`guardar`	`guardar G = resultado`	Nombre base de los archivos del gráfico.
`nomostrar`	`nomostrar G,H`	Prepara los gráficos, pero no los presenta individualmente, por ejemplo si se muestran en un panel.
`eco`, `echo`	`eco G = texto`	Órdenes históricas de mensajes transportadas por el lector; no describen series. Su aceptación léxica no garantiza un efecto de presentación en todas las rutas del motor.

Los nombres simbólicos de v1 se normalizan; los nombres físicos de archivos conservan mayúsculas y minúsculas. No basar un documento nuevo en abreviaturas accidentales del intérprete antiguo. Las órdenes desconocidas o incompletas son errores de lectura.

Ejemplo gráfico independiente, acompañado por una tabla `datos.simul` con columnas T, A y B:

```
PARAPLOT
archivo DATOS = datos.simul
lineas EVOLUCION = DATOS,1:2,3
titulo EVOLUCION = Dos variables contra el tiempo
ejes EVOLUCION = Tiempo(s),Amplitud
rangox EVOLUCION = 0:2
puntos FASE = DATOS,2:3
titulo FASE = B contra A
mostrarjuntos RESUMEN = EVOLUCION,FASE
nomostrar EVOLUCION,FASE
```

4.26.4. 15.3 Órdenes estructuradas (v2)

Orden `.ds`	Regla adicional de v2
`archivo F = datos.simul`	Ruta literal; puede ir entre comillas.
`lineas G = F,1:A,B` / `puntos G = F,T:A`	Nombres simples o índices físicos desde 1; órdenes repetidas acumulan series.
`titulo G = Texto`	Título de gráfico o panel.
`ejes G = Tiempo(s),Temperatura(tempC)`	Etiquetas; la unidad explícita pide conversión real a Tabular2.
`rangox G = 0,10` / `rangoy G = -1,1`	Exactamente dos números finitos y crecientes; coma, no dos puntos.
`mostrarjuntos P = G,H`	Panel vertical, sin repetir gráficos ni anidar paneles.
`puntos G = F,X:Y,limites_y=YMIN:YMAX`	Barras verticales con límites absolutos en columnas de la misma fila.
`lineas G = F,X:Y,limites_x=XMIN:XMAX,limites_y=YMIN:YMAX`	Límites en ambos ejes; con límites se admite una sola serie por orden.

Las palabras clave no distinguen mayúsculas, pero los identificadores de v2 sí. Las referencias adelantadas están permitidas. Los límites de barras deben ser finitos y cumplir inferior $\leq$ centro $\leq$ superior; no son deltas, porcentajes ni desviaciones que Simplot calcule. Todas las columnas involucradas deben tener unidades compatibles. Para temperaturas absolutas usar `tempC`, `tempF` o `K`.

`RESULTADOS`, `UNIDADES` y `ANADIR_GRAFICOS` también se admiten en v2. Las órdenes históricas de selección, funciones, variaciones o `nomostrar` no se convierten automáticamente a v2; se rechazan cuando no están implementadas. La selección de bloques de una tabla con varias corridas, las escalas logarítmicas/temporales, la inversión de ejes, las columnas de panel y los formatos tienen representación en el `.dat` estructurado, pero aún no tienen todas sus órdenes `.ds`. No inventar una palabra clave para esas prestaciones; consultar el contrato gráfico v2.

Ejemplo:

```
RUN_CONTROLS
SIMPLOT_VERSION=[2]
PARAPLOT
archivo DATOS = datos.simul
lineas EVOLUCION = DATOS,T:A,B
titulo EVOLUCION = Evolución
ejes EVOLUCION = Tiempo(s),Amplitud
rangox EVOLUCION = 0,2
puntos FASE = DATOS,A:B
mostrarjuntos RESUMEN = EVOLUCION,FASE
```

4.26.5. 15.4 RESULTADOS, UNIDADES y conversiones

Esta explicación corresponde al procesamiento del problema y se publica también en Siru (`https://www.simusol.org/?Siru#unidades-y-resultados-antes-de-simular`); se retira de las páginas del graficador. La sintaxis se define aquí una sola vez.

Forma abreviada	Pedido común
`TEMP`	Seleccionar la variable conservando sus atributos.

Forma abreviada	Pedido común
`TEMP(Interior)`	Solicitar un nuevo nombre de salida.
`TEMP(Interior,temperatura)`	Declarar además la magnitud.
`TEMP(Interior,temperatura,tempF)`	Solicitar además una unidad de salida, sin redefinir el origen.

Estas secciones son comunes a toda la cadena, no parte exclusiva de PARAPLOT. Su interpretación corresponde al paso `.ds` → `.dat` en Slex, antes de simular. La regla normativa y las prioridades están en §3.9. La conversión solicitada para los resultados se aplica cuando se obtienen sus valores; no se posterga la interpretación del documento hasta Simplot.

```
RESULTADOS
TEMP(Interior,temperatura,tempC),FLUJO(Potencia,potencia,W)
UNIDADES
plantilla=Generico
TEMP:K,temperatura
FLUJO:W,potencia
```

Construcción	Significado
`VARIABLE(nuevo_nombre,magnitud,unidad)`	Solicitud común de resultado; atributos opcionales. El nombre, magnitud y unidad se conservan aunque no se genere un gráfico.
`VARIABLE:unidad,magnitud`	Declaración de origen para el procesamiento común. La prioridad frente al número y FORMAS está en §3.9.
`plantilla=Generico` y reglas por prefijo	Formas históricas que deben conservarse mediante compatibilidad; no autorizan una segunda interpretación física en Simplot.
Metadatos de origen	Deben acompañar los valores preparados, sin reetiquetar un origen ya declarado ni convertir dos veces.
Unidad de salida	Conversión de los resultados solicitada en el documento, no una decisión exclusiva del dibujo.
`RESULTADO GRAFICO`	Sólo previsto, no implementado: excepción posterior a Siru para cambiar unidades de presentación en Simplot, sin alterar el cálculo ni la tabla original. No prioritario.

En el ejemplo, TEMP tiene origen en kelvin y se solicita como resultado en Celsius absoluto. Simplot debe recibir ese resultado ya preparado. Un rótulo «temperatura» por sí solo no transforma una columna ni permite interpretar `C` como Celsius. La implementación histórica todavía efectúa parte de este trabajo en Simplot: es compatibilidad pendiente de migrar, no la distribución definitiva de responsabilidades.

Escritura GNU units	Significado
`tempC`, `tempF`	Temperaturas absolutas, con desplazamiento de origen.
`K`	Kelvin.

Escritura GNU units	Significado
`degC`, `degF`	Intervalos de temperatura; 10 degC = 18 degF = 10 K.
`C`, `F`	Coulomb y farad.
`s`, `min`, `W`, `W/(m^2*K)`	Unidades simples o compuestas. En coeficientes térmicos se usa K o un intervalo, no la función tempC.

Una unidad desconocida, una conversión incompatible o un origen ausente que impida convertir son errores. Los alias de algunos ejemplos antiguos (`seg`, `gradosC`) deben reemplazarse por los nombres de GNU units. La conversión la realizan Tabular2 y Unidades; Simplot no mantiene otra tabla de factores.

4.26.6. 15.5 Un punto, varios estados y ausencia de tiempo

Para el resultado de `INTEGRA=[-1]`, usar `puntos G = F,TEMP:PV`, por ejemplo. No se agrega T ni se duplican muestras para fabricar una curva. En v2 una orden `lineas` con una sola fila también muestra un marcador; en v1 pedir `puntos` explícitamente.

La opción de consola `simplot --pares estado.simul` representa todas las combinaciones distintas de dos columnas. Puede restringirse con `--columnas TEMP,TBH,WWW`; admite entre 2 y 12 columnas. Es una opción del programa, no una sentencia nueva de `.ds`. Las proyecciones de un conjunto de estados no implican una trayectoria temporal.

4.27. 16. Resumen formal, alcance y archivos auxiliares

Esta EBNF describe la estructura del documento; las reglas detalladas de cada sección y sus restricciones semánticas están en §3–15. Las comillas indican literales, las llaves repetición y los corchetes opcionalidad. Cuando aparecen entre comillas, `[` y `]` son caracteres obligatorios del archivo.

```
documento       = { inclusion | seccion | comentario | linea_vacia } [ "END" ]
inclusion       = ( "include" | "input" | "#include" ) ruta
seccion         = encabezado fin_linea { sentencia | comentario | linea_vacia }
sentencia       = contenido ( fin_linea | ";" )
lista_control   = clave "=" "[" valor { "," valor } "]"
incognita       = nombre "=" semilla [ "D:" derivada ]
dato            = nombre "=" expresion [ "D:" derivada ]
residuo         = expresion { marca_ecuacion }
rama_ds         = tipo "-" nombre "," nodo "-" nodo "=" pieza { "&" pieza }
modelo          = tipo "(" cantidad ")" "=" campo { "," campo }
funcion         = nombre "(" argumentos_o_cantidad ")" "=" expresion
salidas         = [ abscisa ":" ] nombre { "," nombre }
orden_grafica   = comando objeto "=" valor_grafico
forma_completa  = nombre TAB forma TAB orientacion TAB numero TAB numero TAB numero
    TAB numero
forma_abreviada = nombre TAB forma
```

El cuerpo de una sección termina al reconocer el próximo encabezado; no lleva llaves. `END` termina la fuente actual. `TAB` significa el carácter de tabulación. La regla de sentencia con `;` es de `.ds`; en `.d` ese carácter inicia un comentario. Las gramáticas ejecutables separan el reconocimiento léxico de la interpretación de cada sección y de la construcción del sistema; una EBNF abreviada no reemplaza sus validaciones.

Familia	Comprobaciones que no basta expresar en la EBNF
Nombres	Declaraciones disponibles, ausencia de ramas repetidas, incógnitas y ecuaciones compatibles.
Circuito	Tipo existente, cantidad de valores del modelo, masa GND, topología y estados admisibles.
Expresiones	Operadores y funciones de ExprTk, dominio numérico, semillas que permitan evaluarlas.
Integración	Una familia compatible, IDs existentes, estados y derivadas, condiciones de continuación, paso y tolerancias.
Dibujo	Campos de FORMAS y números finitos; una sección vacía conserva significado.
Gráficos	Fuentes y columnas existentes, versión, unidades, rangos y límites coherentes.
Inclusiones y recursos	Rutas válidas y recursos externos disponibles al leer o ejecutar.

La referencia corresponde al lector de este directorio. Sus fuentes de sintaxis son `d2datdinc.slex`, `parametros.y`, `ecuacion.y`, `declaraciones.y`, `control.y`, `data.y`, `edef.y`, `piezas.y`, `rama.y`, `salidas.y`, `sceptre.y`, `expresion.y` y `comas.y`; `dato.c` y `userfunc.c` implementan interpretación y compatibilidad. El catálogo de ramas es `biblioteca/libraryori.siru`; los catálogos numéricos están junto a él. El espejo web se genera desde este documento, no se mantiene una segunda gramática escrita a mano.

4.28. 17. Ejemplos completos y comprobaciones

Descargar los archivos de cada fila en un mismo directorio. Los fragmentos de las secciones anteriores explican reglas; estos archivos sí son entradas completas. Trabajar sobre una copia, pues Siru escribe tablas, registros y, cuando corresponde, esquemas junto a la entrada.

Ejemplo	Archivos	Qué comprobar
Sólo Calcula / ExprTk	expresion.ds (`../../d2datdinc/doc/ejemplos/expresion.ds`)	`siru -nosimplot expresion.ds`: un punto con X=2; sin columna de tiempo artificial.
Función Fortran	fortran.ds (`../../d2datdinc/doc/ejemplos/fortran.ds`), cuadrado.f90 (`../../d2datdinc/doc/ejemplos/cuadrado.f90`)	La misma ecuación y resultado mediante una función externa; requiere habilitación y compilador de §9.
RC en `.d`	rc.d (`../../d2datdinc/doc/ejemplos/rc.d`)	Fuente, resistencia no lineal y capacitor; integración temporal.

Ejemplo	Archivos	Qué comprobar
RC en `.ds` e inclusión	rc.ds (`../../d2datdinc/doc/ejemplos/rc.ds`), biblioteca.siru (`../../d2datdinc/doc/ejemplos/biblioteca.siru`)	El mismo modelo con tipos explícitos y biblioteca relativa.
Ecuaciones diferenciales sin ramas	oscilador.ds (`../../d2datdinc/doc/ejemplos/oscilador.ds`)	Y1≈cos(T), Y2≈−sin(T), sin nodos ni GND; método del trapecio con paso 0.01.
Modelo propio	modelo-propio.ds (`../../d2datdinc/doc/ejemplos/modelo-propio.ds`)	Resistencia cúbica: IR1=2 para VR1=8 y coeficiente 1.
Subcircuito `.d`	subcircuito.d (`../../d2datdinc/doc/ejemplos/subcircuito.d`)	Modelo DIVISOR con terminales y nombres interiores; VR2T1=5.
Tabla incorporada	tabla.ds (`../../d2datdinc/doc/ejemplos/tabla.ds`)	Interpolación de la tabla interna: X=0.5, Y=1.
Partición IMEX	imex.ds (`../../d2datdinc/doc/ejemplos/imex.ds`)	X≈exp(−11T), ARS222 con paso 0.01.
FORMAS vacía	automatico.ds (`../../d2datdinc/doc/ejemplos/automatico.ds`)	Solicita esquema automático y resuelve un punto.
Símbolos eléctricos	divisor.ds (`../../d2datdinc/doc/ejemplos/divisor.ds`)	Divisor con sus formas incorporadas; tabuladores conservados.
Símbolos térmicos	termico.ds (`../../d2datdinc/doc/ejemplos/termico.ds`)	Analogía térmica y registros de ramas y nodos.
PARAPLOT histórico	graficos-v1.ds (`../../d2datdinc/doc/ejemplos/graficos-v1.ds`), datos.simul (`../../d2datdinc/doc/ejemplos/datos.simul`)	`simplot -graf=4 graficos-v1.ds`: evolución, plano de fase y panel.
PARAPLOT estructurado	graficos-v2.ds (`../../d2datdinc/doc/ejemplos/graficos-v2.ds`), datos.simul (`../../d2datdinc/doc/ejemplos/datos.simul`)	Los mismos datos con versión 2; no requiere Calcula.

Para revisar sólo la lectura, `d2datdinc -v ejemplo.ds -o ejemplo.dat`. Para comprobar el cálculo, ejecutar Siru; un DAT válido no garantiza que el sistema tenga solución ni que converja desde cualquier semilla. Para los ejemplos gráficos ejecutar Simplot, pues las columnas y unidades se comprueban al leer las tablas.

4.28.1. Fuentes y mantenimiento de la referencia

Material	Función
README del lector (`../../d2datdinc/README.md`)	Semántica, compatibilidad y pruebas de d2datdinc.
Manual de Siru (`../../../siru/doc/MANUAL.md`)	Ejecución, integradores, análisis y diagnósticos.
Contrato PARAPLOT v1 (`../../../simplot/doc/DAT-PARAPLOT.md`)	Transporte de órdenes y responsabilidades del lector y graficador.
Contrato gráfico v2 (`../../../simplot/doc/DAT-SIMPLOT-V2.md`)	Fuentes, series, unidades, paneles y capacidades del DAT.
IMEX (`../../../siru/doc/IMEX.md`), ROS (`../../../siru/doc/ROS.md`), GLM (`../../../siru/doc/GLM_CONTRATO.md`)	Contratos y restricciones de cada familia.
Documentación oficial de ExprTk (`https://www.partow.net/programming/exprtk/index.html`)	Lenguaje de expresiones y funciones de la biblioteca de Arash Partow.

La fuente de esta página está en `slex/dsa/doc/GRAMATICA.tex`; su generador, configuración de publicación y espejo están en `sitio/gramatica-ds`. Cuando cambie el lector o el consumidor de una sección, actualizar su sintaxis, sus límites y el ejemplo correspondiente, regenerar el espejo y comprobarlo antes de publicar.

4.29. Motor y salida de Maxima (2026-11-06)

`MOTOR = nombre` selecciona `siru`, `siru-scipy`, `siru-sceptre`, `siru-spice`, `spice`, `maxima`, `sceptresimusolnuevo`, `sceptrenovender` o `sceptresimusolviejo`. `scipy` permanece como alias de `siru-scipy`. Los recorridos combinados con SCEPTRE y SPICE todavía no tienen adaptador de intervalos y se rechazan explícitamente. En M/DU se escribe en el nivel superior del DU, fuera de los bloques. En el documento histórico se escribe fuera de PARAPLOT. El lector conserva `GLOBAL.motor`. El lanzador usa Siru si falta la selección; `--motor` prevalece sobre el documento.

Dentro de PARAPLOT, `maxima = formulas` pide soluciones analíticas para Gnuplot y `maxima = tablas` pide muestreo numérico (predeterminado). Otros motores ignoran esta orden. Un bloque que solo contiene esa orden conserva los gráficos implícitos. En DU se cierra con FIN. No se admiten pedidos contradictorios de ambos modos en una misma corrida. Contrato, ejemplos y límites: ANALITICAS.md (`../../../simplot/doc/ANALITICAS.md`).

4.29.1. Informe HTML de PARAPLOT

`html = si` pide a Simplot guardar un informe HTML autocontenido con modelo, datos, resultados y gráficos. Los textos se presentan en cuadros copiables. `html = no` o su ausencia no lo pide. La orden no altera las ecuaciones ni el motor. En DU pertenece al bloque PARAPLOT, cerrado por FIN. Contrato: HTML.md (`../../../simplot/doc/HTML.md`).

4.29.2. Selección explícita y predeterminada de integración

En un .d sin rutina explícita, el lector conserva la selección histórica INTEGRA=[216] para Siru y la marca `implicito` en el control del DAT. El lanzador no interpreta esa marca como una elección hecha para otro motor. Una declaración INTEGRA explícita se valida contra el diccionario del motor antes de ejecutar; por ahora el SCEPTRE genérico traduce 112 a TRAP y 216 a XPO. Spice admite 112, traducido a `method=trap`. Las demás traducciones se rechazan hasta estar implementadas.

5. Ecuaciones con retardo en `.dsa`

Estado: implementada en el lector y el conductor temporal de Siru. Los dos archivos `.dsa` de ejemplo al final se pueden ejecutar. La sintaxis de esta página aún es experimental; los límites que no se implementaron se indican de forma explícita.

Actualización 11/11/2026: esta página describe el conductor histórico 226. También están implementados 282, BS23 adaptativo (`../../../siru/doc/RETARDO_ADAPTATIVO.md`), y 283, DP54, retardos variables y neutrales (`../../../siru/doc/RETARDO_DP54.md`). El lector de M/DU es C y la ruta nativa no usa Python ni `.d`/`.ds` intermedios. SciPy permanece como motor externo Python independiente.

5.1. Alcance inicial

La primera versión describiría ecuaciones diferenciales con retardos constantes positivos:

$$\dot{x}(t) = f\big(t, x(t), x(t-\tau_1), \ldots, x(t-\tau_m)\big).$$

En 226 quedan fuera los retardos dependientes del tiempo o del estado, las ecuaciones neutrales (retardos de derivadas), los pasos adaptativos y la coordinación de circuitos con llaves, lazos o cortes. El paso debe cumplir `SIRU_PASO <= min(retardos)`; se rechaza la corrida si un dato hace que no se cumpla. Las capacidades de 282 y 283 tienen contratos propios, enlazados arriba.

5.2. Sintaxis

Dentro de `MODELO`, `RETARDOS` asigna a cada alias el estado y el retardo que representa. El alias se usa en las ecuaciones como cualquier otra variable algebraica. `HISTORIA` define, para cada estado, su función prescrita antes del tiempo inicial. Las expresiones pueden depender de `T` y de parámetros del modelo.

```
MODELO
  ESTADOS: X
  PARAMETROS: A, B, TAU
  ECUACIONES
    DX + A*X - B*X_RET = 0
  FIN
  RETARDOS
    X_RET = X, TAU
  FIN
  HISTORIA
    X = 1
  FIN
FIN
```

La forma de `RETARDOS` es `ALIAS = ESTADO, RETARDO`. El alias debe ser único, no puede coincidir con un estado o parámetro, y cada referencia debe tener una declaración. Cada estado debe tener una ecuación de deriva y una expresión de historia. La cantidad de estados de la historia debe coincidir con la del modelo.

Los valores de los retardos pueden depender de parámetros fijados por cada caso DU, pero una vez iniciado el caso se mantienen constantes. Los retardos deben ser finitos y estrictamente positivos. El tiempo inicial y final y `SIRU_PASO` pertenecen al DU; el valor inicial del estado debe coincidir con la historia evaluada en el tiempo inicial.

5.3. Ejemplo 1: un retardo

La ecuación $x'(t) = x(t-1)$, con historia $x(t) = 1$ para $t \le 0$, se escribiría así:

```
DESCRIPCION: "Ecuación escalar con un retardo constante"
MODELO
  ESTADOS: X
  ECUACIONES
    DX - X_RET = 0
  FIN
  RETARDOS
    X_RET = X, 1
  FIN
  HISTORIA
    X = 1
  FIN
FIN
```

El DU ejecutable está en retardo-simple.DU.dsa (`../../../siru/doc/ejemplos-retardados/retardo-simple.DU.dsa`):

```
INTEGRA=[226]
DATOS
  X = 1
  T = 0
  TIEMPOFINAL = 3
FIN
RESULTADOS
  X
FIN
USOS
  CONTROLES
    SIRU_PASO = [0.05]
    SIRU_RETARDO_INTERPOLACION = ["lineal"]
  FIN
FIN
```

El selector `INTEGRA=[226]` ejecuta RK4 por método de pasos. Con paso `0.05`, el ejemplo produce `X(2)=3.500000000000002` y `X(3)=6.166875`; la solución exacta en `t=3` es `37/6`. La interpolación lineal y los quiebres propagados explican el error observado con ese paso.

5.4. Ejemplo 2: varios retardos y estados

Cada retardo tendría un alias propio, incluso cuando dos alias refieran al mismo estado. La historia se declara una vez por estado:

```
MODELO
  ESTADOS: X, Y
  PARAMETROS: A, B, C, TAU1, TAU2
  ECUACIONES
    DX + A*X - B*X_T1 - C*Y_T2 = 0
    DY + A*Y - B*X_T2 = 0
  FIN
  RETARDOS
    X_T1 = X, TAU1
    X_T2 = X, TAU2
    Y_T2 = Y, TAU2
  FIN
  HISTORIA
    X = 1
    Y = exp(T)
  FIN
```

```
FIN
```

El ejemplo ejecutable está en retardos-dos-estados.M.dsa (`../../../siru/doc/ejemplos-retardados/retardos-dos-estados.M.dsa`) y retardos-dos-estados.DU.dsa (`../../../siru/doc/ejemplos-retardados/retardos-dos-estados.DU.dsa`). Exige `TAU1>0` y `TAU2>0`; el paso no puede superar el menor de ambos valores. La interpolación Hermite cúbica usa los estados y sus derivadas en los extremos de cada intervalo aceptado; la opción lineal es la predeterminada. Las discontinuidades iniciales y las propagadas por los retardos pueden reducir el orden observado, por lo que el usuario debe elegir el paso en consecuencia.

Desde la raíz del árbol de Siru se ejecutan así:

```
PERL5LIB=siru/lib perl siru/bin/siru --modelo siru/doc/ejemplos-retardados/retardo-
    simple.M.dsa --salida /tmp/retardo-simple.simul < siru/doc/ejemplos-retardados/
    retardo-simple.DU.dsa
PERL5LIB=siru/lib perl siru/bin/siru --modelo siru/doc/ejemplos-retardados/retardos-
    dos-estados.M.dsa --salida /tmp/retardos-dos-estados.simul < siru/doc/ejemplos-
    retardados/retardos-dos-estados.DU.dsa
```

5.5. Separación entre M y DU

El modelo M contendría estados, ecuaciones de deriva, alias retardados, duraciones de retardo y funciones de historia. El DU contendría los valores iniciales compatibles, los parámetros de cada caso, el tiempo final, el paso, la interpolación y los resultados solicitados. Repetir distintos casos DU reutilizaría la estructura del modelo, pero cada caso tendría su propia historia numérica.

El lector valida la estructura y los nombres sin requerir que estén presentes los datos de ejecución. El conductor temporal valida los valores, la positividad y finitud de los retardos, la compatibilidad del valor inicial con la historia y la restricción del paso antes de abrir la salida.

5.6. Selector y controles implementados

Tema	Comportamiento implementado
Interpolación	Lineal por omisión; Hermite cúbica optativa.
Paso	Fijo y no mayor que el retardo positivo mínimo del caso.
Historia	Una expresión por estado, evaluada para `T` anterior al inicio.
Método	RK4 dentro del método de pasos, selector global `INTEGRA=[226]`.
Salida	Puntos aceptados del estado; no salida densa ni adaptación.
Casos múltiples DU	Una historia y una integración independientes por caso; salida `.simul`.

Se rechazan la mezcla con Monte Carlo de parámetros, la adaptación y la salida densa. Para desactivar explícitamente una adaptación heredada se admite `SIRU_ADAPT=[0]`.

5.7. Condiciones de parada — 07/11/2026

INTEGRA=[226] admite condiciones escalares del DU. Con SIRU_EVENTOS=[0] comprueba el predicado en las fronteras aceptadas; con [1] localiza contactos y primeros cruces ocultos mediante reintegración del mismo método de pasos.

Parte	Tratamiento
Estados	Cada prueba RK4 parte del mismo estado y derivada confirmados.
Historia	Lee los mismos puntos confirmados y la función HISTORIA anterior al inicio. No registra sondeos ni los utiliza como pasado de otras pruebas.
Paso máximo	Conserva h<=min(retardos). Cada consulta de una etapa accede a un pasado ya conocido; no extrapola al futuro del extremo confirmado.
Interpolación	Conserva lineal o Hermite cúbica, seleccionada en el caso. La localización no eleva el orden de la aproximación histórica.
Predicado	Puede usar estados, algebraicas del punto de Calcula y alias de valores retardados. Usa las funciones escalares interpretadas del mismo modelo.
Salida	Publica solo el inicial y extremos aceptados. La historia contiene pasos+1 registros; los sondeos no consumen pasos.
Condición inicialmente falsa	Publica el inicial compatible y no integra. Un nombre desconocido o expresión no admitida produce error en vez de omitir la condición.
Varios DU	Cada caso tiene estado, condición e historia propios.
Diagnóstico	Paradas RETARDOS informa consultas, intervalos, parada terminal y registros de historia por caso.
Continuación/adaptación	Esta conexión no añade continuación guardada ni adaptación al conductor 226. Los conductores adaptativos 282/283 tienen contratos propios enlazados al comienzo.

Ejemplo de DU (`../../../siru/ejemplos/eventos-parada/retardo.DU.dsa`), ejecutable con `ejemplos/dae-retardo/modelo.M.dsa` sin --dae-residual. Para $X'(t)=X(t-1)$, historia $X=1$ y $Z=X^2$, se verifica también una parada sobre X_RET en t=1.237: consulta historia calculada después del inicio, no solo la función anterior al origen.

`tests/retardado_eventos.t` verifica contactos, cruces ocultos, estados, algebraicas, alias retardados, ambas interpolaciones, dos DU, errores e historia no contaminada. El núcleo anterior conserva sus pruebas de solución por tramos, convergencia y acceso a historia. Se mantienen los límites de cobertura finita del localizador (`../../../siru/doc/EVENTOS_PARADA.md`).

Validación: 74 comprobaciones nuevas de eventos y 29 del núcleo previo, 103 aprobadas. El cierre conjunto con el intérprete, Calcula, la variable E en una fuente real y la matriz por número totaliza 1.869 comprobaciones dirigidas, sin ventanas de gráficos.

6. DSA como flujo entre programas

6.1. Contrato acordado

Canal	Contenido	Regla
Entrada estándar	Modelo, datos y operación en formato DSA, concatenables en ese orden.	Siru lee un flujo completo sin exigir nombres de archivo para el modelo o los casos.
Salida estándar	DSA completo `X.dsa` o resultado `X.R.dsa`, según la elección declarada en `PARAPLOT`.	Debe poder leerlo otro programa DSA; no es una impresión de diagnóstico ni un `.simul` con otra extensión.
Error estándar	Diagnósticos, progreso y errores.	No contamina el resultado redirigible.

El concepto rector es **filtro**: `productor | siru | consumidor`. La entrada puede formarse con `cat M.dsa DU.dsa | siru --dsa`. `PARAPLOT COMPLETO` conserva M y DU y agrega R en `X.dsa`; `PARAPLOT R` emite solo R, con simulación y PARAPLOT, en `X.R.dsa`. Si no hay PARAPLOT, se usa la salida R. El nombre del archivo lo da la redirección de la salida estándar. Tanto el modelo como los datos pueden provenir de otros filtros. Una salida con varios casos o una trayectoria conserva dentro del DSA sus casos, variables, unidades cuando correspondan y valores, sin depender del nombre del archivo para interpretarse.

Si la entrada contiene dos casos DU para el modelo, la salida contiene dos secciones `SIMUL`, una por caso y en el mismo orden. Cada sección `SIMUL` lleva su propio título identificador dentro del DSA, tanto en la salida completa como en la reducida. Si el caso declara un título, se conserva; si no, el título es `Caso 1`, `Caso 2`, etc., según el orden de los casos en el flujo de entrada. La numeración se reinicia en cada ejecución del filtro y no depende de archivos temporales ni del método de integración. No se confunden las dos simulaciones en una única tabla anónima. La sintaxis para declarar un título propio del caso aún debe definirse en el lenguaje DSA.

6.2. Estado de la implementación al 11 de noviembre de 2026

Parte	Estado actual	Cambio pendiente
Lectura DSA	`siru --dsa` toma M y DU concatenados de entrada estándar. El lector nativo en C los procesa sin construir `.d` o `.ds` intermedios.	Mantener pruebas de ida y vuelta con las demás familias.
Resultado temporal	El filtro entrega `SIMUL` titulados por salida estándar; `.simul` es temporal interno.	Definir cómo declara cada caso un título propio.
Mensajes	El filtro dirige los mensajes de ejecución al error estándar.	Comprobar también los diagnósticos de las familias todavía no probadas en este modo.
Compatibilidad	Los caminos existentes con archivos nombrados y `.simul` siguen disponibles.	Extender la cobertura del filtro sin atribuirle capacidades antes de probarlas.

La interfaz `siru --dsa` está implementada para un modelo seguido de casos DU. `PARAPLOT R` emite solo secciones `SIMUL` tituladas y el bloque PARAPLOT; `PARAPLOT COMPLETO` conserva la entrada M+DU y agrega las secciones SIMUL. Se usa, por ejemplo, `cat M.dsa DU.dsa |`

`siru --dsa > R.dsa` o `> X.dsa`, según el modo declarado en PARAPLOT. Cada línea de la tabla `.simul` se representa como cadena DSA dentro de su sección; Slex puede volver a leerla. El título propio declarado en un caso aún no tiene sintaxis definida; mientras tanto se aplica `Caso N`. El modo filtro ejecuta los DU por el mismo conductor temporal de Siru y mantiene `.simul` solo como producto interno temporal.

7. De SPICE al .dsa: lo que traduce cir2dsa

`cir2dsa CIRCUITO.cir` (o `d2datdinc CIRCUITO.cir --dsa BASE`) lee un netlist de SPICE y escribe el par `BASE.M.dsa` y `BASE.DU.dsa`, para que el circuito lo corra siru. La regla es estricta: lo que no se sabe llevar a siru **corta**, diciendo cada cosa con su renglón; nada se saltea en silencio. La referencia es ngspice 47.

7.1. El circuito

En SPICE	En el .dsa
`R`, `C`, `L`, con valor, cuenta o `.model`	`Resistencia`, `Condensador`, `Bobina`. `tc1`, `tc2`, `tce`, `temp` y `dtemp` multiplican el valor según la temperatura; una R infinita queda de 1e30 ohm.
`V`, `I` con `DC`, `PULSE`, `SIN`, `PWL`, `EXP`, `SFFM`	Fuentes cuyo valor es la forma de onda escrita con `if()` y `mod()`; siru reconoce los codos de un pulso como puntos de quiebre.
`TRNOISE`, `TRRANDOM`	El azar (§7.4): la secuencia se genera al traducir y queda como una `TABLA` que la fuente recorre.
`E`, `G`, `H`, `F`, `B`	Fuentes cuyo valor es una cuenta con las tensiones y corrientes del circuito. `ddt(x)` pasa tal cual (§4.6.1); `idt(x)` corta.
`S` con `.model sw (vt vh ron roff)`	Llave resistiva (`LLAVES_RESISTIVAS`); con `vh`, la histéresis con `anterior()`.
`D` con `.model d`	El diodo de ngspice: la corriente por tramos (directa, inversa y, con `BV`, ruptura), `RS` en serie y `GMIN` en paralelo, como una resistencia de 1/GMIN (así una fuente de corriente que alimenta diodos no deja un corte hecho solo de fuentes de corriente, que siru no admite); la carga de juntura (`CJO`, `VJ`, `M`, `FC`) y de difusión (`TT`) en un `CondensadorIncremental` en paralelo, que en un barrido `.dc` solo no va (en continua no conduce). $V_t = kT/q$ con las constantes de ngspice; la corriente de saturación depende de la temperatura (la del circuito o `temp=` de la instancia, contra `TNOM`); la capacidad de juntura fuera de `TNOM` corta. Con `thermal` y `RTH0`, el autocalentamiento: la temperatura de la unión es la del dispositivo más la tensión del nodo térmico, que recibe la potencia de la unión y de `RS` y tiene `RTH0` y `CTH0` a masa (`diode-soa-simple.cir`: el diodo sin autocalentamiento a $2 \cdot 10^{-6}$ de ngspice; el que se calienta, a 1,7 % en el codo, donde la temperatura sube 370 °C). `@d1[gd]`, `[cd]`, `[qd]` e `[id]` son salidas. Contra ngspice 47 (`diode_ac_par.cir`, de -5 a 0,1 V): conductancia y capacidad a $3 \cdot 10^{-4}$, la corriente del barrido a $3 \cdot 10^{-8}$.

En SPICE	En el .dsa
`X` y `.subckt`	Se aplastan; el nodo `n` de la instancia `x1` queda `n_x1`. Los `PARAMS:` pueden ser cuentas sin llaves (`beta=20e3/stime`); una `.func` del cuerpo usa los parámetros de cada instancia (queda `f1_x1`); en una expresión, un número del cuerpo es un número (no un puerto) salvo dentro de `v(...)`, e `i(V1)` del cuerpo es `i(V1_x1)`.
`.param`, `.func`, `.temp`	Parámetros de `DATOS`, funciones expandidas (también en las cuentas que se hacen al traducir), la temperatura. `ic={valor}` de un C o una L, con llaves, es su condición inicial; los argumentos de una forma de onda entre comillas o llaves (`sin(0 'vmax' '1/stime')`) son una cuenta cada uno. Un parámetro de modelo entre comillas (`is='1e-12*temper'`) se calcula.
`.if`, `.elseif`, `.else`, `.endif`	Se resuelven al leer, con los `.param` numéricos: quedan los renglones de la rama que vale.
Nombres	Sin distinguir mayúsculas; un modelo puede empezar con un dígito (`1n4002`).

7.2. Los análisis

En SPICE	En el .dsa
`.tran tstep tstop [tstart [tmax]] [uic]`	`INTEGRA=[112]`, `SIRU_PASO`, `SIRU_PASO_MAX`, el paso por error de truncamiento (`TRTOL`, `CHGTOL`, `ABSTOL`) y `SALIDA_DESDE`. Sin `uic` parte del punto de operación (`SCEPTRE_IC=[1]`).
`.op`	El punto de operación, una fila.
`.dc FUENTE a b inc`	Una rampa de la fuente a lo largo de la corrida, a paso fijo: un punto por valor del barrido.
`.dc temp a b inc`, `set temp`	Una corrida por temperatura.
`.ac dec|oct|lin N f1 f2`	El análisis en frecuencia de siru; la fuente con `ac` es la excitación, y una R con `ac=` usa `AC_MODO`.
`.pz in+ in- out+ out- vol pz`	`POLOS_CEROS (V(out) / fuente)`: la entrada es la fuente de tensión entre `in+` e `in-`; siru deja los polos y ceros en `NOMBRE.pz`. La transferencia de corriente (`cur`) corta.
`.noise v(sal) fuente dec N f1 f2`	`RUIDO`: cada resistencia es una fuente de ruido térmico, una corriente en paralelo de densidad 4kT/R; siru deja `onoise_spectrum` e `inoise_spectrum` en `NOMBRE.ruido`. Contra ngspice, a $5 \cdot 10^{-7}$. Solo el ruido térmico de las resistencias: el de disparo y el $1/f$ de los diodos todavía no.
`.ic v(nodo)=valor`	El valor inicial de los condensadores que tocan el nodo, con `IC_DADAS`.
`.options`	`reltol`, `abstol`, `chgtol`, `trtol`, `method`, `maxord`; las de listado se aceptan; `vntol` corta.

En SPICE	En el .dsa
`.print`, `.plot`	`RESULTADOS`. `v(n)`, `v(a,b)`, `i(V)`, `v1#branch`, `all`, `allv` (las tensiones), `alli` (las corrientes), `tran1.v(1)` (el nombre de la corrida adelante: aquí cada corrida es su archivo), `xmem.x1` (el nodo `x1` de la instancia `xmem`, que queda `x1_xmem`), `i(v.x1.v1)` (la fuente `v1` de la instancia `x1`, `v1_x1`), un nodo suelto, y una cuenta (`i(v1)-i(v2)`), que queda como una variable `SAL_K` con su ecuación. Las opciones del dibujo se ignoran.
`.measure tran`	`MEDIDAS` (§4.19): `MAX`, `MIN`, `PP`, `AVG`, `RMS`, `INTEG`, `MAX_AT`, `MIN_AT`, `FIND`, `DERIV`, `WHEN`, `TRIG/TARG`, `param=`. Un nombre repetido queda `nombre_2`. Una medida que ngspice rechaza (función o parámetro desconocido, falta el `=`, `TRIG` sin `TARG` o sin `rise/fall/cross`, vector mal escrito, `meas sp` sin ese análisis) queda de tipo `falla` con su motivo; sin nombre o con un análisis desconocido, ngspice ni la crea y no queda nada. Sin `VAL`, 0, como ngspice.

7.3. El guion (.control)

El bloque `.control` de ngspice es un programa. Se traduce lo que tiene equivalente: los análisis, las medidas y las comprobaciones. Lo que solo muestra algo (`echo`, `plot`, `listing`, `set` de pantalla) se acepta; lo demás corta nombrando el comando.

En el guion	En el .dsa
`tran`, `op`, `dc`, `ac`, `meas`, `print`	Lo mismo que sus tarjetas.
`let x = cuenta`	Una medida de tipo `cuenta`. Con `v()` o `i()`: tras `op` es un número; tras `ac`, el complejo `U3_ac`; tras `tran` es una curva, otro nombre de esa cuenta. Volver a dar un nombre deja `x_2`.
`define f(a,b) cuenta`	Se expande en cada llamada.
`if ... echo "ERROR" ... end`	Una verificación (`VERIFICACIONES`) de que no se dé la condición que lleva al error.
`if` con condición vacía, nombre inexistente o `$?variable`	Se resuelve al traducir, como lo hace ngspice.
`if C / let x=a / else / let x=b`	`x = C ? a : b`.
`foreach`, `while`, `dowhile`, `repeat`	Se ejecutan al traducir, si su lista o su condición son números: el cuerpo queda escrito una vez por vuelta.
`$variable`, `{$variable}`, `$&vector`, `set nombre = texto`	Se cambian por su valor.
`compose x values a b c`, `vecmax()`, `vecmin()`	Listas de cuentas; el mayor y el menor.
`@R2[i]`, `@modelo[param]`, `@v1[dc][k]`, `@v1[pulse]`	La corriente de un elemento, el valor de un parámetro, un punto de un barrido, los números de una forma de onda.
`alter`, `alterparam`, `altermod` entre dos análisis	Cada análisis es una corrida aparte, con los cambios hechos hasta ahí: `BASE`, `BASE-2`, `BASE-3`...

En el guion	En el .dsa
`source archivo`	Carga ese circuito (`$inputdir` o un camino relativo, desde la carpeta del `.cir`); con una sola carga y sin `setcirc`. `cutout` es de los gráficos.
`stop when time = T`, `alter`, `resume`	El circuito con esos valores cambiando en T: una fuente V o I de continua pasa a fuente B (`time <= T ? viejo : nuevo`), una R con `tc1`/`tc2` a `r = time <= T ? ...` con la temperatura ya puesta. `time` en una cuenta del guion es la escala de la corrida (`tiempo`).
El guion que trata las curvas como vectores	Cuando un `let` usa `length()`, `fft()` o `ifft()`, o se le pone un índice a un `v()` o un `i()` (`v(1)[i]`), lo que sigue al análisis pasa entero al bloque `GUION` (§4.22), que siru corre sobre la tabla: `let`, `let x[k] =`, `compose`, `while`, `repeat`, `if`, `echo`; `v(n)`, `i(x)` y los nodos sueltos son columnas; `@modelo[par]` y `@r1[r]`, su número; `@d1[id]`, su columna; `linearize`, con el paso del `.tran`. `FFT_Leakage.cir`: la distorsión, 111,05 % en los dos (ngspice da el signo contrario, que sale del redondeo de una raíz compleja). `diode-soa-sh.cir`: la zona segura, igual donde no depende de la temperatura del diodo.
`mtimeavg()`, `deriv()`, `integ()`	`CURVAS` (§4.21). `length()`, `K*unitvec(N)` y `let x = "vec"` sobre las listas.
`linearize`, `fft v(1)`, `psd N v(1)`	`ESPECTROS` (§4.20); el `psd` da lo mismo que ngspice. `let x = mag(v(1))` después de un `fft` es del espectro.
Después de un `ac`: `v(2)`, `frequency`	Uno por frecuencia: una cuenta con ellos es una lista, y `vecmax()` toma la mayor. `pi`, `i`, `boltz` y `echarge` valen.
Después de un `pz`: `pole(k)`, `zero(k)`	El polo o el cero `k` (`polo_k`, `cero_k` en las medidas).

7.4. El azar

Las fuentes de ruido de ngspice sacan números al azar mientras corre la simulación. Acá la secuencia se genera **al traducir** y queda escrita en el `.dsa` como una `TABLA` de pares (tiempo, valor) que la fuente recorre con `Table()`: siru no tira números, y la corrida se repite igual cuantas veces se haga. La semilla es `.options seed=N` (1 si no se da); con otra semilla sale otra secuencia.

En SPICE	La tabla
`TRNOISE(NA NT ...)`	Ruido blanco: un valor gaussiano de desvío `NA` cada `NT`, el primero 0, y entre uno y otro la recta.
`TRNOISE(NA NT NALPHA NAMP)`	Ruido $1/f^{\mathrm{NALPHA}}$ de amplitud `NAMP`: ruido blanco pasado por el filtro $h_k = h_{k-1}(\mathrm{NALPHA}/2 + k - 1)/k$, en ronda.
`TRNOISE(... RTSAM RTSCAPT RTSEMT)`	Telégrafo: un escalón de altura `RTSAM` que sube tras un tiempo exponencial de media `RTSCAPT` y baja tras otro de media `RTSEMT`.

En SPICE	La tabla
`TRRANDOM(TIPO TS TD P1 P2)`	Un valor nuevo cada `TS` desde `TD`, que se mantiene hasta el siguiente: 1 uniforme en ±`P1` más `P2`; 2 gaussiano de desvío `P1` y media `P2`; 3 exponencial de media `P1` más `P2`; 4 Poisson de media `P1` más `P2`.

No da lo mismo que ngspice punto a punto, y no puede: ngspice usa su propio generador, lo siembra con el reloj si no se le fija la semilla, y pide los números en el orden en que avanza su corrida. Lo que se compara es la estadística. Contra ngspice 47, con 1000 muestras (18/11/2026): ruido blanco de desvío 1, 0,996 (ngspice 0,986); uniforme en ±1, desvío 0,580 (0,570); exponencial de media 0,5, media 0,496 (0,492); gaussiano de media 1 y desvío 2, 0,99 y 1,99 (1,10 y 1,98); Poisson de media 3, 3,02 (2,91); telégrafo, fracción del tiempo arriba 0,68 (0,72; lo esperado, 0,67). En el ruido $1/f$ el espectro tiene la misma pendiente (-1) en los dos, pero el desvío de ngspice es mayor (1,74 contra 1,19): ngspice, en su versión con FFTW, multiplica las dos transformadas tomando los elementos de a uno en vez de a pares (`1-f-code.c`), y eso le saca la media a la secuencia y le sube la amplitud; hecha la cuenta de las dos maneras, la suya da 1,83 y la del filtro bien aplicado 1,41, que es la de acá. Se deja la cuenta correcta y no se copia la de ngspice (decisión del autor, 18/11/2026).

7.5. Lo que todavía corta

Monte Carlo (`mc_source`, `setseed`), el análisis `.pss`, el ruido de los diodos, los transistores, `idt()`, y del guion `cutout`, `stop`/`resume`, `setcirc` y los vectores de más de una dimensión.

8. Quiero construir y utilizar un modelo: qué escribir y cómo entregarlo

Esta guía parte de la tarea del autor: construir un modelo, darle datos y pedir una operación. Son tres partes separables, aunque un programa requiera reunirlas en un solo archivo. Los ejemplos usan la sintaxis actual; las interfaces comunes todavía propuestas se identifican expresamente. Revisión: 27 de octubre de 2026.

8.1. 1. Las tres cosas que tengo que proporcionar

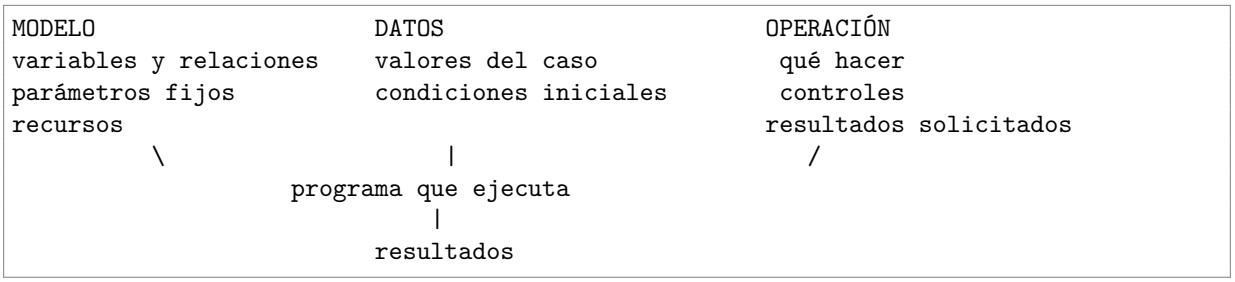
Parte	Pregunta que debo responder	Qué escribo
Modelo	¿Qué sistema describo?	Variables, ecuaciones —incluidas las del circuito y del método de integración—, parámetros fijos y recursos.
Datos	¿Qué caso quiero calcular?	Valores de las variables que se suministran para cada uso del modelo; fijan el punto que se calcula o las condiciones iniciales de una evolución.
Operación	¿Qué quiero hacer con ese modelo y esos datos?	Resolver un punto, buscar un equilibrio, simular, barrer parámetros u otro análisis; opciones de ejecución.

RESULTADOS queda fuera del modelo. Selecciona qué variables informar en cada uso, con unidades opcionales, sin crear variables ni agregar ecuaciones; puede incluir parámetros conocidos. Cambiar esa selección no cambia el modelo. Fijar un valor corresponde a DATOS, no a RESULTADOS. La ubicación histórica de pedidos de salida en los archivos existentes no modifica esta separación conceptual.

El modelo no es su estado inicial. Puedo usar el mismo modelo con muchos conjuntos de datos. Un punto que Calcula resuelve puede ser el estado inicial de una simulación, un estado posterior conocido o un caso independiente de cualquier trayectoria.

El método de integración es parte del modelo. Las ecuaciones del sistema y las relaciones del método numérico —RK, GLM u otro— constituyen el modelo utilizado para calcular la evolución. Los datos aportan los valores del caso y la información inicial necesaria; la operación indica qué hacer con ese modelo y esos datos. Se puede cambiar el método de integración conservando las mismas leyes físicas. Que la sintaxis actual coloque su selector en `RUN_CONTROLS` no cambia esta clasificación conceptual.

Separación conceptual:



8.2. 2. Primero escribo el modelo del sistema

En esta guía, «modelo» sin otra calificación se refiere al **modelo del sistema completo**. Un **modelo de elemento** describe las variables y ecuaciones de un tipo de elemento y se aplica a cada rama de ese tipo. Son conceptos diferentes: los modelos de elementos contribuyen a construir el modelo del sistema.

El modelo reúne cuatro partes. Las variables y ecuaciones pueden declararse directamente o generarse a partir de otras descripciones, como las ramas de un circuito o un método de integración.

Parte del modelo	Qué contiene	Ejemplo
a. Variables	Las declaradas explícitamente, las que aparecen al definir las ramas y conexiones de un circuito, y las derivadas. Se identifican por su nombre, significado, unidades y convenciones de signo; para una derivada se indica de qué variable es y respecto de cuál.	V, I, X; DX=dX/dT. Una rama puede introducir su tensión y corriente sin que haya que declararlas nuevamente.

Parte del modelo	Qué contiene	Ejemplo
b. Ecuaciones	Las añadidas explícitamente, las relaciones que definen las derivadas, las leyes y relaciones de conexión generadas por el circuito, y las ecuaciones del método de integración cuando se incorpora uno.	$V-R \cdot I=0$; $DX+K \cdot X=0$; balances de corrientes; relaciones entre etapas y estados de un RK o GLM.
c. Parámetros fijos	Valores que se fijan como parte del modelo y se conservan entre sus usos. Si se quiere cambiar uno por caso, debe tratarse como dato de entrada de ese modelo parametrizado.	Una resistencia $R=3$ ohm mantenida en todas las evaluaciones, o una constante física utilizada en sus ecuaciones.
d. Recursos	Funciones, tablas, bibliotecas y archivos necesarios para definir o evaluar las relaciones.	Una tabla de propiedades o una biblioteca de modelos de elementos.

Las variables tienen valores; el modelo establece cómo se relacionan, qué valores permanecen fijos y cuáles se aportan en cada uso. Un valor conservado dentro del modelo se declara como parámetro fijo. Un valor que se entrega para un caso concreto es un dato. Los valores de las incógnitas se obtienen al resolver las ecuaciones con esos parámetros y datos. Una variable solicitada como resultado puede ser conocida o calculada; pedir que se informe no añade una ecuación.

La presencia de derivadas no obliga a calcular una dinámica. Por ejemplo, la relación $DX+K \cdot X=0$ permite calcular $DX=-6$ en un único punto con $K=2$ y $X=3$. Para calcular una evolución temporal deben existir las relaciones que determinan las derivadas y pedirse explícitamente la evolución durante un intervalo de tiempo, con sus condiciones iniciales y controles de integración. Esa operación calcula estados en distintos instantes. Dar un tiempo de evaluación para un solo punto no pide una trayectoria; tampoco una lista de casos independientes constituye por sí sola una evolución temporal.

La presencia de un circuito implica, a través de sus ramas y conexiones, variables y ecuaciones adicionales en el modelo del sistema. Cada rama tiene un nombre, conecta nodos y pertenece a un tipo de elemento. Ese tipo tiene un modelo de elemento que define sus variables y relaciones. Al declarar una rama, se aplica el modelo de su tipo con los nombres y parámetros de esa rama; así se incorporan sus variables y ecuaciones al sistema. Las conexiones aportan además relaciones de conservación y compatibilidad, como el balance de corrientes en los nodos y la relación de las tensiones de rama con los potenciales de nodo.

Concepto	Qué representa	Ejemplo
Modelo del sistema	El conjunto de variables, ecuaciones, parámetros fijos y recursos del problema completo.	Un circuito RC, junto con las ecuaciones adicionales que se hayan declarado.

Concepto	Qué representa	Ejemplo
Tipo de elemento y su modelo	Una definición reutilizable de las variables y relaciones que aporta un elemento de ese tipo.	Un tipo resistivo cuyo modelo relaciona tensión, corriente y resistencia.
Rama del circuito	Una aplicación concreta de un tipo de elemento, con nombre, nodos y parámetros propios.	R1 y R2 usan el mismo modelo de elemento resistivo, pero cada una introduce sus propias variables y ecuaciones.

El generador reúne las variables y ecuaciones provenientes de las ramas y conexiones con las declaradas explícitamente. No hace falta repetir a mano las que ya genera el circuito. La presencia de un circuito no obliga a calcular una evolución temporal: puede resolverse un único punto del sistema. Si describo el sistema completo mediante variables y ecuaciones explícitas, no necesito declarar un circuito.

8.2.1. Dónde escribo cada cosa en .ds

Contenido del modelo	Sección o declaración actual
Relaciones explícitas	`EQUATIONS`: cada expresión es un residuo que debe anularse.
Incógnitas	`VARIABLES`: el valor asignado es una semilla numérica, no un dato impuesto.
Estado y derivada	Asociación `X=... D:DX` en `DEFINED_PARAMETERS`; declarar DX y la ecuación que la determina. La asociación pertenece al modelo; el valor de X pertenece a los datos.
Parámetro fijo o dato incluido en el archivo	`DEFINED_PARAMETERS`, con un número, puede transportar ambos en la sintaxis actual. Hay que distinguir su función: valor conservado en el modelo o valor suministrado para ese uso.
Relación auxiliar	Por ejemplo `POT=V*I` en `DEFINED_PARAMETERS`: aunque esté allí, es una relación, no un dato independiente.
Tipos de elemento y modelos de elementos	`ELEMENT_DESCRIPTION` / `ELEMENT_DEFINITION`, o biblioteca incluida: define las variables y relaciones que aporta cada tipo.
Ramas y conexiones del circuito	`ELEMENTS`: cada rama indica su nombre, tipo de elemento, nodos y parámetros. La aplicación de los modelos de elementos y las conexiones genera variables y ecuaciones del sistema.
Funciones	`FUNCTIONS`, tablas utilizadas por las ecuaciones y recursos externos admitidos.
Constantes fijadas	`CONSTANTS`; no tratarlas automáticamente como variables libres del caso.

Contenido del modelo	Sección o declaración actual
Resultados solicitados	`OUTPUTS` y los pedidos admitidos de `RESULTADOS`: identifican lo que se quiere informar.

La sintaxis actual puede mezclar modelo y datos en una misma declaración. La separación que buscamos es de significado; no requiere inventar encabezados que el lector todavía no entiende.

8.3. 3. Después doy los datos del caso

Los datos se suministran para cada uso, separados conceptualmente de la definición del modelo. Pueden darse por una entrada independiente o incluirse en el mismo archivo en un bloque destinado al caso. En un cálculo de un punto son los valores conocidos que lo determinan; en una simulación aportan, entre otros, los valores iniciales. Compartir archivo no convierte esos datos en parámetros fijos del modelo.

Dato que proporciono	Qué significa
Valor conocido de una variable	Restricción que debe satisfacer la solución del punto.
Valor que puede cambiar entre usos	Resistencia, masa, conductividad u otra magnitud que el modelo admite como dato, en lugar de conservarla como parámetro fijo.
Estado inicial	Valor desde el que comienza una evolución, sujeto a la consistencia de las ecuaciones.
Tiempo de evaluación	Necesario si las relaciones dependen explícitamente del tiempo, incluso al calcular un solo punto.
Semilla del solver	Aproximación para buscar la solución; es una opción numérica, no un dato físico impuesto.
Historia de un método	Datos numéricos adicionales para continuar un método con memoria. No son otra ley física ni siempre se deducen del último estado.

Debo aportar suficientes datos independientes para determinar las incógnitas. Puedo cambiar cuáles son las variables conocidas si la formulación y el motor lo permiten. Esto no garantiza solución única: hay que comprobar consistencia, dominio de validez y convergencia.

Los datos pueden estar en el mismo `.ds` que el modelo o entregarse por separado. El llamador actual de Calcula admite líneas externas y un bloque comentado `CALCULA DATOS`. Esas líneas son casos sucesivos; no constituyen por sí solas una política general de reemplazo de todas las declaraciones del archivo. En esa interfaz, los valores iniciales incluidos en el archivo pueden completar datos faltantes según los grados de libertad. Esa conducta del programa no elimina la distinción entre parámetros fijos y datos del caso.

8.4. 4. Por último pido una operación

Quiero...	Operación	Información adicional
Conocer las variables restantes en un estado	Resolver un punto	Variables conocidas; semillas adecuadas si hacen falta.

Quiero...	Operación	Información adicional
Encontrar un estado estacionario	Buscar equilibrio o punto de operación	Restricciones estacionarias apropiadas. No basta llamar «inicial» al punto.
Seguir el cambio de los estados	Simular una evolución	Condiciones iniciales, intervalo, modelo de integración, paso y tolerancias.
Retomar una simulación	Continuar una evolución	Estado y memoria que requiera el método.
Estudiar frecuencia, sensibilidad o incertidumbre	Análisis específico	Su definición, excitaciones, variables y parámetros; motor que lo soporte.
Ver los resultados	Solicitar tablas y gráficos	Variables, unidades y representación.

La repetición de casos es una tarea aparte. Primero se define qué se hace con un caso: resolver su punto, buscar un equilibrio o calcular la evolución que parte de sus condiciones iniciales. Repetir esa operación con otros datos significa ejecutar otros casos; no forma parte de la definición de un punto ni de sus consecuencias.

Organización de varios casos	Cómo se ejecuta	Qué significa
Varias llamadas	Llamar a Calcula, Siru o SCEPTRE una vez por caso, con los datos correspondientes.	Cada llamada realiza la operación solicitada para ese caso.
Una llamada múltiple	Entregar varios conjuntos de datos a una interfaz que admita procesarlos en una sola llamada.	Agrupa ejecuciones de casos; puede reutilizar el modelo preparado, sin cambiar el significado de cada cálculo.
Barrido de casos	Generar conjuntos de datos variando uno o varios valores y ejecutar la operación elegida para cada conjunto.	Es una forma de organizar la repetición, mediante varias llamadas o una llamada múltiple.

Un caso puede producir un único punto o toda una trayectoria si se pidió evolución temporal. Los puntos de esa trayectoria están vinculados por las ecuaciones y la integración; no equivalen a repetir cálculos con datos independientes. A la inversa, procesar muchos casos en una sola llamada no los convierte en una evolución temporal. La disponibilidad de una llamada múltiple depende de la interfaz de cada programa; no se supone una sintaxis común ya implementada.

El catálogo común expresa la intención; cada programa tiene hoy su propia entrada y cobertura. No convertir silenciosamente un barrido estático en una excitación temporal, ni un punto arbitrario en un equilibrio.

8.4.1. 4.1 Si voy a simular: uso el método de integración del modelo

La operación «simular» utiliza el método de integración definido como parte del modelo. Puedo seleccionar un método existente; definir uno nuevo exige describir sus ecuaciones, coeficientes, arranque y memoria, además de comprobar sus propiedades.

Parte del método	Runge–Kutta (RK)	General Linear Method (GLM)
Definición	Tabla A, b, c que relaciona etapas y estado siguiente.	Bloques A, U, B, V y c que relacionan etapas y memoria de entrada/salida.
Información de entrada	Estado actual, tiempo, paso y parámetros.	Memoria externa, tiempo, paso y parámetros; puede incluir estados o derivadas de pasos anteriores.
Arranque	El primer estado debe satisfacer el modelo; las etapas implícitas pueden necesitar semillas.	Debe existir una regla para obtener la historia que aún no está disponible.
Adaptación	Estimador de error y política de aceptación si se usa paso adaptativo.	Estimador y política compatibles con el cambio de paso y la actualización de memoria.
Continuación	Conservar el estado y cualquier información que el motor necesite.	Conservar también la historia o efectuar un rearranque declarado.

Para una ecuación explícita y'=f(t,y), la estructura conceptual es:

```
RK:
  Y_i = y_n + h Σ_j A_ij f(t_n+c_j h, Y_j)
  y_(n+1) = y_n + h Σ_j b_j f(t_n+c_j h, Y_j)

GLM:
  Y = h A F + U H_n
  H_(n+1) = h B F + V H_n
```

Y representa las etapas; F, sus derivadas; H, la memoria externa del método. En un GLM, H puede contener cantidades escaladas según el método. En Siru, la generación puede combinar estas relaciones con el sistema residual, sin exigir que el usuario despeje previamente todas las derivadas. Eso no garantiza que cualquier sistema algebraico-diferencial sea resoluble por cualquier método.

8.4.2. 4.2 Qué escribo hoy para elegir un método

Los siguientes son selectores alternativos de `RUN_CONTROLS`; no deben copiarse todos juntos.

Elección	Declaración en .ds	Qué más debo proporcionar
Un punto, sin integración	`RK=[-1]`	Datos del punto. El uso de RK en este selector es una convención de la interfaz actual.
RK clásico de cuatro etapas	`RK=[7]`	Estado inicial, intervalo y control de paso.
Regla trapezoidal implícita	`RK=[12]`	Los mismos datos, más una solución viable de las etapas implícitas.

Elección	Declaración en .ds	Qué más debo proporcionar
Euler mediante el motor GLM	`GLM=[1]`	Estado inicial y controles.
BDF2 directo	`GLM=[21]`	Estado inicial; el método implementado arranca con Euler implícito.
BDF2 por datos efectivos	`GLM=[20]`	Controles compatibles con esa representación.
Adams–Bashforth 2	`GLM=[30]`	Estado inicial; arranque implementado con Euler explícito.
IMEX ARS(2,2,2)	`PGLM=[2]`	Partición `IMEX_EXPLICITA` para los estados y paso fijo.
Rosenbrock ROS2	`ROS=[1]`	Controles y tratamiento del Jacobiano según el contrato ROS.
Composición Strang	`SPLIT=[2]`	Mapas de subflujos y declaraciones de autonomía/exactitud, dentro de sus límites actuales.

Los números son identificadores de cada catálogo, no órdenes del método ni una numeración universal. En particular, `GLM=[1]` y `RK=[1]` no seleccionan lo mismo.

Ejemplo de controles para un paso fijo:

```
RUN_CONTROLS
GLM=[21]
SIRU_PASO=[0.01]
SIRU_ADAPT=[0]
```

Para adaptación, debo proporcionar tolerancias, límites y un estimador admitido. Los GLM actuales tienen una política específica de arranque y comparación; no son automáticamente BDF de orden variable. Debo consultar el contrato del método antes de combinarlo con llaves, cambios topológicos, discontinuidades o continuaciones.

8.4.3. 4.3 Si quiero definir un método propio

Tengo que escribir	Dónde está la implementación actual de referencia
Coeficientes RK y metadatos de la tabla	rk.siru (`../../d2datdinc/biblioteca/rk.siru`).
Definición GLM, memoria, bloques, arranque y capacidades	glm.json (`../../d2datdinc/biblioteca/glm.json`) y contrato GLM (`../../../siru/doc/GLM_CONTRATO.md`).
Partición explícita/implícita	Contrato IMEX (`../../../siru/doc/IMEX.md`).
Jacobiano, escalas y estimación para ROS	Contrato ROS (`../../../siru/doc/ROS.md`).

Seleccionar un ID usa una definición existente; escribir un número nuevo no crea un método. Para incorporar uno hay que definirlo en el catálogo y verificar que el generador y el motor

admitan su estructura. Comprobar dimensiones de matrices no demuestra orden, consistencia ni estabilidad. Una definición GLM arbitraria, una memoria distinta o una tabla RK nueva pueden requerir desarrollo adicional.

8.5. 5. Cómo entrego las tres partes a cada programa

Programa	Modelo	Datos	Operación
Calcula	Sistema instalado, módulo, `.din`, o `.d`/`.ds` mediante Siru.	Línea de datos, archivo, entrada estándar o datos incluidos en la fuente.	Resolver puntos; la interfaz directa `.d`/`.ds` usa el llamador de puntos.
Siru	`.d`/`.ds` y sus recursos.	Parámetros y estados en el archivo; el modo `--calcula-puntos` admite datos externos.	Controles del archivo; `--solo-punto` fuerza el punto inicial sin evolución.
SCEPTRE	`.d` de SCEPTRE; `.ds` por la traducción existente y sus límites.	Parámetros, valores de componentes y condiciones del deck.	Instrucciones de `RUN CONTROLS` y análisis admitidos.
Puyuspa	Sistema del catálogo o archivo cargado en una página que lo admita.	Formulario del sistema; condiciones del archivo para simular; datos adicionales para puntos en las fuentes nuevas.	Botón o formulario que llama al motor correspondiente.
SPICE/ngspice	Netlist, modelos de elementos o dispositivos, subcircuitos y bibliotecas.	Valores, parámetros y condiciones según la operación.	Directivas de análisis del dialecto: `.op`, `.tran`, `.ac`, `.dc`, etc.

Separación conceptual no significa que hoy todos reciban tres archivos independientes. SCEPTRE y SPICE pueden reunir las tres partes en su archivo de entrada. La futura interfaz común debe ensamblarlas o traducirlas sin cambiar su significado.

8.5.1. 5.1 Calcula: modelo más archivo de casos

Si guardé el modelo como `modelo.ds` y mis conjuntos de datos en `datos.txt`:

```
calcula modelo.ds datos.txt
```

Cada línea de datos corresponde a un caso. Para un modelo que declare `j1` y `r2`, por ejemplo:

```
j1=2 A,r2=3 ohm
j1=3 A,r2=4 ohm
```

La misma intención puede entregarse al llamador:

```
siru --calcula-puntos modelo.ds --datos-archivo datos.txt
```

Estas entradas requieren las versiones que incorporan el llamador. No son una orden de integrar aunque la fuente contenga un método RK o GLM.

8.5.2. 5.2 Siru: ejemplo completo con modelo, datos y operación identificados

Guardo lo siguiente como `decaimiento.ds`. Las magnitudes de este ejemplo son adimensionales; los comentarios identifican las tres partes sin introducir sintaxis nueva.

```
CIRCUIT_DESCRIPTION
Decaimiento: DX = -K*X

# MODELO: derivada y relacion fisica
VARIABLES
DX=0
EQUATIONS
DX+K*X

# DATOS: parametro y estado inicial
# D:DX asocia el estado con su derivada: esa parte es modelo
DEFINED_PARAMETERS
K=1
X=1 D:DX
T=0

# OPERACION: intervalo de la simulacion
TIEMPOFINAL=2
RUN_CONTROLS
# MODELO: metodo de integracion
RK=[7]
# CONTROLES DE EJECUCION
SIRU_PASO=[0.01]
SIRU_ADAPT=[0]
CONTROL_EQUATIONS
T<TIEMPOFINAL

# RESULTADOS SOLICITADOS
OUTPUTS
T:X,DX
END
```

Quiero hacer	Qué entrego
Simular	`siru decaimiento.ds -nosimplot`
Calcular solo el punto inicial	`siru --solo-punto decaimiento.ds -nosimplot`
Usar el mismo modelo con otro caso	`siru --calcula-puntos decaimiento.ds --linea 'k=2,x=3'`
Simular con BDF2 directo	Sustituir únicamente `RK=[7]` por `GLM=[21]` y ejecutar Siru.

En esta interfaz, Calcula recibe los nombres generados en minúsculas. El tiempo no aparece como dato independiente del punto en este modelo autónomo. Para el punto K=2, X=3, la relación exige DX=−6; no hay que integrar para calcular esa derivada. Para la trayectoria inicial K=1, X=1, la solución de referencia es X=exp(−T). Cambiar el integrador no cambia esa ley.

8.5.3. 5.3 SCEPTRE: las tres partes reunidas en un .d

Ejemplo de un circuito RC; los rótulos de la tabla explican las partes, no son nuevos encabezados.

Parte	Dónde la pongo
Modelo	Ramas y conexiones en `ELEMENTS`, con los modelos de elementos nativos; modelos de elementos o funciones adicionales si hacen falta.
Datos	Valores de las ramas, parámetros y `INITIAL CONDITIONS`.
Operación	`RUN CONTROLS`: rutina, tiempo final y análisis; `OUTPUTS` para solicitar resultados.

```
CIRCUIT DESCRIPTION
RC alimentado por una fuente de 1 V
ELEMENTS
E1, GND-1 = 1
R1, 1-2 = 1
C1, 2-GND = 1
INITIAL CONDITIONS
QC1 = 0
OUTPUTS
VC1, IR1
RUN CONTROLS
INTEGRATION ROUTINE = TRAP
STOP TIME = 2
END
```

Se entrega como `sc1 circuito.d`. Para `sc1 modelo.ds` interviene la traducción: hay que revisar sus avisos, pues no toda ecuación libre o método de Siru tiene equivalente. No se debe asumir que `GLM=[21]` se ejecutará como BDF2 en SCEPTRE por haber escrito ese selector en la fuente.

8.5.4. 5.4 Puyuspa: dónde cargar el archivo

Página pública de simulaciones: `https://simusol.org/puyuspa/simular.html`.

1. En **Un archivo propio**, pulsar el selector que dice **Sin archivos seleccionados**.
2. Seleccionar el archivo `.d` o `.ds` —hasta 200 KB—.
3. Pulsar **Subir y simular**. El archivo aporta modelo, datos iniciales e instrucciones de simulación.

RLC es un ejemplo del catálogo; no limita los archivos propios que se pueden seleccionar. La página de vapor húmedo es un sistema particular de Calcula, no el selector de archivos.

Interfaz	Qué permite y dónde están los datos
Página pública descrita por el usuario el 26/10/2026	Cargar `.d`/`.ds` y simular con los datos y controles del archivo.
`simular.html` en las fuentes locales nuevas	Además permite calcular puntos y escribir datos adicionales por línea. Su presencia en las fuentes no prueba que esté publicada.
`modelos.html` en las fuentes locales nuevas	Carga solo `.d`; permite pedir puntos o simulación. El cuadro de datos adicionales se envía al cálculo de puntos, no a la simulación.

Interfaz	Qué permite y dónde están los datos
Sistemas binarios y `.din` registrados	Se seleccionan del catálogo instalado; no se cargan como ejecutables desde el navegador.

No hay que prometer todavía tres controles web independientes y generales para modelo, datos y operación. Esa es la organización que debe guiar la evolución de la interfaz.

8.5.5. 5.5 SPICE/ngspice: modelo, datos y análisis en una netlist

Parte	Escritura habitual en el dialecto ngspice
Modelo del sistema	Componentes y nodos, modelos de elementos (`.model`), subcircuitos (`.subckt`), funciones y bibliotecas.
Datos	Valores de componentes y fuentes, `.param`, condiciones iniciales apropiadas para el análisis.
Operación	`.op` para punto de operación, `.tran` para evolución, `.ac` para análisis de pequeña señal, `.dc` para barrido DC.
Opciones numéricas	Método, tolerancias y demás opciones admitidas por el motor. No usar los IDs RK/GLM de Siru como si fueran opciones SPICE.

Ejemplo nativo de un divisor resistivo, guardado como `divisor.cir`:

```
Divisor resistivo: modelo, datos y operacion
* DATOS DEL CASO
.param RA=1000 RB=1000 VS=10
* MODELO PARAMETRIZADO: componentes y conexiones
V1 in 0 DC {VS}
R1 in out {RA}
R2 out 0 {RB}
* OPERACION
.op
.end
```

Se ejecuta con `ngspice -b divisor.cir`. El punto de operación debe dar v(out)=5 V. En este ejemplo algebraico la operación es sencilla; en un circuito dinámico, calcular `.op` no equivale a imponer un estado inicial arbitrario. Las condiciones iniciales y semillas deben interpretarse según el análisis solicitado.

La entrada directa `.d`/`.ds` para una versión adaptada de ngspice es una propuesta, no una función que esta guía dé por instalada. Existe un traductor parcial en la dirección inversa, SPICE → `.ds`, llamado `cir2ds`; sus equivalencias y omisiones deben revisarse. En particular, traducir un barrido a una rampa temporal no conserva en general el significado de un barrido estático.

8.6. 6. Antes de ejecutar, compruebo que entregué todo

Pregunta	Debe quedar explícito
¿Está completo el modelo?	Variables, relaciones, conexiones, recursos accesibles y método de integración si se va a calcular una evolución.
¿Qué datos fijan este caso?	Valores conocidos, unidades y condiciones iniciales separadas de las semillas.
¿Qué quiero calcular?	Punto, equilibrio, evolución u otro análisis concreto.
¿Cómo se hará numéricamente?	Si integro: definición o selección RK/GLM/etc., arranque, memoria, paso y error; si resuelvo un punto: opciones del solver que correspondan.
¿Puede hacerlo el programa elegido?	Capacidades, restricciones y traducciones necesarias, sin omisiones ocultas.
¿Qué resultado espero recibir?	Variables, tablas, unidades y gráficos solicitados.
¿Cómo verificaré el resultado?	Un caso conocido, balances, residuos o comparación con una solución de referencia.

8.7. 7. Nuevo lenguaje: archivos .dsa

La extensión acordada para el nuevo formato es `.dsa`. Se reserva para el lenguaje que separa explícitamente modelo, datos y operación. Los archivos `.d` y `.ds` conservan su sintaxis existente: renombrarlos no los convierte al nuevo formato. Los ejemplos de las secciones anteriores describen esas interfaces actuales.

El diseño propuesto para `.dsa` se organiza así:

Bloque	Significado
`MODELO`	Modelo del sistema: variables, derivadas, ecuaciones, circuito, parámetros fijos, integración y recursos.
`MODELO_ELEMENTO`	Definición reutilizable de un tipo de elemento, distinta del modelo del sistema.
`DATOS`	Valores impuestos para un caso, asociados al modelo correspondiente.
`OPERACION`	Qué hacer con un modelo y unos datos: resolver un punto o pedir explícitamente una evolución u otro análisis.
`SEMILLAS`	Aproximaciones numéricas para el solver; nunca se confunden con datos impuestos ni con la declaración de variables.

Los bloques de modelo, datos y operación pueden reunirse en un archivo `.dsa` o distribuirse en archivos distintos. La sintaxis exacta de las referencias entre archivos todavía debe especificarse.

En este lenguaje, declarar `DX` en `VARIABLES` no le asigna cero ni ningún otro valor. Su asociación con `derivada(X, tiempo)` pertenece a `DERIVADAS`; una igualdad como `DX = -K * X` pertenece a `ECUACIONES`; un valor fijo como `K = 2`, a `PARAMETROS`; el dato `X = 3`, al caso en `DATOS`; y una aproximación inicial `DX = 0`, si se necesita, a `SEMILLAS`. Las ecuaciones se escriben como igualdades, sin obligar al autor a convertirlas en residuos.

Seleccionar RK o GLM en `INTEGRACION` incorpora el método al modelo, pero no ordena ejecutar una trayectoria. La operación de evolución debe solicitarse explícitamente. La repetición de casos sigue siendo una tarea exterior a la operación de cada caso.

Estado: nombre y organización conceptual acordados; gramática detallada, lector y adaptación a los programas todavía por implementar. No se anuncia aquí que Calcula, Siru, SCEPTRE, Puyuspa o SPICE acepten ya `.dsa`.

8.8. Referencias para escribir y ejecutar

- Gramática .d/.ds: sintaxis concreta y ejemplos.
- Manual de Siru (`../../../siru/doc/MANUAL.md`): ejecución y límites de las operaciones.
- Contrato GLM (`../../../siru/doc/GLM_CONTRATO.md`), IMEX (`../../../siru/doc/IMEX.md`) y ROS (`../../../siru/doc/ROS.md`): modelos numéricos.
- Calcula (`../../../../calcula/calcula-6.0/README`) y Puyuspa (`../../../../calcula/puyuspa-7.0/DOCUMENTACION.md`): interfaces actuales.
- Entrada de SCEPTRE (`../../../sceptre2novender/pasos/POLITICA-ENTRADA`): interpretación y traducción.
- SPICE (`../../../spice.md`) y cir2ds (`../../examples/11/README.md`): formatos y alcance del traductor.
- Modelos compartidos (`../../../MODELOS-COMPARTIDOS.md`): arquitectura y catálogo común de operaciones.