

Package: trama (via r-universe)

October 19, 2026

Title Interactive Node-Graph Computation with Live Previews

Version 0.5.23

Description Build and run computation graphs interactively. Nodes are typed R functions with declared ports and parameters; a flow is a plain JSON document; execution is pull-based, content-addressed and incremental, so editing a parameter recomputes only what depends on it. Every node renders its own result in the canvas with a registered renderer, making each step of a pipeline visible rather than implied. Domain knowledge lives entirely in registrable collections, never in the core, so new domains are added without modifying the package.

License MIT + file LICENSE

URL <https://github.com/uaipedro/trama>

BugReports <https://github.com/uaipedro/trama/issues>

Copyright file inst/COPYRIGHTS

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R (>= 4.1), dplyr, lubridate

Imports stats, rlang, jsonlite, htmltools, shiny, later, curl, httpuv

Suggests testthat (>= 3.0.0), withr, mirai, pkgload

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake make libuv1-dev libssl-dev zlib1g-dev

Repository <https://uaipedro.r-universe.dev>

Date/Publication 2026-10-19 19:33:14 UTC

RemoteUrl <https://github.com/uaipedro/trama>

RemoteRef HEAD

RemoteSha acbb8f4e290406545c13907d4af06e810bfaef24

Contents

.tr_presentation_ops	4
tr_adapter	5
tr_adapter_for	5
tr_add	5
tr_app	6
tr_bust	7
tr_catalog	7
tr_catalog_json	7
tr_category	8
tr_cli	9
tr_collection	10
tr_compatible	11
tr_dataset	12
tr_dataset_load	13
tr_datasets	13
tr_dependency_collection	14
tr_dependency_core	14
tr_dependency_importmap	14
tr_doc	15
tr_doc_apply	15
tr_doc_json	15
tr_doc_migrate	16
tr_doc_parse	16
tr_doc_read	17
tr_doc_subset	17
tr_doc_terminals	17
tr_doc_validate	18
tr_doc_write	18
tr_errors	19
tr_executor_pool	19
tr_executor_sequential	19
tr_export_code	20
tr_flow	21
tr_flow_code	22
tr_flow_doc	22
tr_fn	23
tr_galeria_captura	23
tr_galeria_itens	24
tr_galeria_sincronizar	25
tr_get_node	25
tr_get_type	26
tr_handle_failed	26
tr_help_test_card	26
tr_icon	27
tr_is_template	27
tr_link	27

tr_mcp	28
tr_node	28
tr_node_raiox	30
tr_op_semantic	31
tr_param	31
tr_param_bool	31
tr_param_col	32
tr_param_enum	33
tr_param_int	33
tr_param_num	34
tr_param_text	34
tr_param_theme	34
tr_plan	35
tr_plan_blocked	35
tr_plan_keys	36
tr_plan_pending	36
tr_plugins	36
tr_port	37
tr_port_default	37
tr_port_free	38
tr_pressuposto	38
tr_preview	38
tr_progress	39
tr_project	39
tr_project_at	39
tr_project_flow	40
tr_project_gc	40
tr_project_new	41
tr_project_save	41
tr_project_set_galeria	42
tr_project_set_marca	42
tr_project_set_sugestoes	43
tr_project_set_themes	43
tr_provenance	44
tr_ref	45
tr_registry	45
tr_registry_for	46
tr_replay	46
tr_retry	46
tr_run	48
tr_run_logger	49
tr_run_plan	49
tr_runs	49
tr_scheduler	50
tr_server	50
tr_set	51
tr_store	51
tr_store_gc	52

tr_store_get	52
tr_store_handle	53
tr_store_has	53
tr_store_path	53
tr_store_put	54
tr_store_put_error	54
tr_stream_command	55
tr_submit	55
tr_template	56
tr_template_dir	56
tr_template_json	56
tr_template_list	57
tr_template_op	57
tr_template_parse	57
tr_template_read	58
tr_template_save	58
tr_test	58
tr_test_table	59
tr_test_type	60
tr_text	60
tr_theme	61
tr_theme_names	62
tr_transitions_read	62
tr_type	63
tr_ui	64
tr_use	64
tr_value	65
tr_when	65
tr_when_padrao	66
Index	67

.tr_presentation_ops *Ops que mexem SÓ em apresentação. Todo o resto é semântico.*

Description

A lista é essa direção de propósito. Enumerar as ops semânticas (e tratar o resto como cosmético) faria o custo de esquecer uma op nova ser exatamente o bug que R/session.R diz ter erradicado: edição aceita, revisão avança, preview não atualiza, nenhum erro. Enumerando as cosméticas, o custo de esquecer vira "recomputou à toa" — barulhento e inofensivo.

Usage

.tr_presentation_ops

`tr_adapter`*Adaptador de tipo: conversão registrada entre dois tipos.*

Description

Inserido pelo motor NA ARESTA (vira marca visual, não nó) e entra na chave de cache. É o que dispensa nós identidade como o `height_as_field` do insumo, mantendo o motor cego a semântica de domínio.

Usage

```
tr_adapter(from, to, fn)
```

`tr_adapter_for`*Busca o adaptador from_type -> to_type, ou NULL se não houver (tipos iguais não precisam de adaptador).*

Description

Busca o adaptador from_type -> to_type, ou NULL se não houver (tipos iguais não precisam de adaptador).

Usage

```
tr_adapter_for(from_type, to_type, registry = .tr_default_registry)
```

`tr_add`*Acrescenta um nó. ... são params (nomeados). from liga a saída de um ou mais nós existentes às portas de entrada obrigatórias ainda livres deste, na ordem declarada — o caso comum de "encadeia" sem precisar nomear porta.*

Description

Acrescenta um nó. ... são params (nomeados). from liga a saída de um ou mais nós existentes às portas de entrada obrigatórias ainda livres deste, na ordem declarada — o caso comum de "encadeia" sem precisar nomear porta.

Usage

```
tr_add(
  flow,
  id,
  type,
  ...,
  from = NULL,
  label = NULL,
  seed = NULL,
  position = NULL
)
```

tr_app

*Sobe o editor.***Description**

Os assets do front vêm de `inst/www` via `htmltools::htmlDependency()`, e cada coleção carregada publica os próprios em `tr-<id>/`. A ordem de carga é determinística e importa: `runtime.js` cria `window.tr` (os registros), as coleções registram `renderers` e `widgets`, e só então o editor monta.

Usage

```
tr_app(
  project = tr_project("."),
  flow = "main",
  executor = tr_executor_sequential(),
  port = getOption("trama.port", tr_port_default()),
  controle = getOption("trama.controle", TRUE),
  ...
)
```

Arguments

port	Porta em que o editor sobe. O padrão é fixo (e não sorteado pelo Shiny) para que a URL do editor seja sempre a mesma; se estiver ocupada, <code>tr_app()</code> procura a próxima livre e anuncia no console. Mude o padrão da sessão com <code>options(trama.port = ...)</code> , ou passe <code>port = NULL</code> para deixar o Shiny sortear.
controle	Sobe o canal de controle para agentes (<code>trama-agente</code> na linha de comando; ver <code>vignette/site "Agentes"</code>): servidor em <code>127.0.0.1</code> com token gravado em <code>.trama/control-agente.json</code> . Desligue com <code>options(trama.controle = FALSE)</code> .

tr_bust	<i>Invalida por prefixo de coleção — válvula para o gap de dependência externa.</i>
---------	---

Description

Casa se QUALQUER coleção que produziu o artefato for a pedida. Nó comum tem uma só, derivada do node_type. A região de fluxo tem o conjunto dos membros, gravado no handle, porque o node_type dela é trama/stream_region e lia como coleção "trama": tr_bust(store, "minha_colecao") não a alcançava, todo nó comum recomputava, e o histórico de dez mil pontos continuava vindo do código de antes do upgrade — calado, pra sempre.

Usage

```
tr_bust(store, collection = NULL)
```

tr_catalog	<i>Catálogo do registro em forma serializável: tipos, categorias, nós e adaptadores — o que o front usa pra desenhar a paleta e validar conexões sem round-trip.</i>
------------	--

Description

Catálogo do registro em forma serializável: tipos, categorias, nós e adaptadores — o que o front usa pra desenhar a paleta e validar conexões sem round-trip.

Usage

```
tr_catalog(registry = .tr_default_registry)
```

tr_catalog_json	<i>tr_catalog() como JSON.</i>
-----------------	--------------------------------

Description

tr_catalog() como JSON.

Usage

```
tr_catalog_json(registry = .tr_default_registry)
```

tr_category

Categoria: agrupamento na paleta e cor do cabeçalho dos cards.

Description

A cor deve vir do PAPEL do bloco no fluxo, não da coleção: assim um ajuste de modelo e um ARIMA têm a mesma cor em qualquer coleção, e o tipo do que o bloco produz fica nas portas. Os papéis seguem a ordem em que o analista trabalha, e são genéricos, nada de domínio:

Usage

```
tr_category(id, label, color = "#6366f1", role = NULL)
```

Arguments

id	Identificador da categoria, único no registro.
label	Rótulo na paleta.
color	Cor fixa, usada só quando role não é dado.
role	Papel dos blocos da categoria no fluxo; um dos listados acima.

Details

- origem: traz dados para o fluxo (ler arquivo, dados de exemplo).
- preparacao: limpa, transforma ou reorganiza.
- inspecao: descreve o que chegou (resumo, gráfico do dado bruto).
- ajuste: constrói o objeto de análise (modelo, decomposição, PCA).
- leitura: lê um ajuste que já existe (quadro da ANOVA, coeficientes, médias, previsões, cargas). São os mostradores do modelo, não um passo a mais da sequência; por isso a cor é uma versão clara da do ajuste.
- avaliacao: julga um resultado (teste, pressupostos, métricas).
- saida: tira algo do fluxo (gravar arquivo).

O papel da categoria é o padrão dos blocos dela; um bloco que faz outra coisa declara o próprio em [tr_node\(\)](#). A cor de cada papel é do editor e muda com o tema. color fica para quem não declara papel: vale nos dois temas, e a tinta do título é escura.

tr_cli

*Linha de comando do canal de controle.***Description**

Fala com o editor aberto (tr_app()) pelo servidor de R/control.R. Toda saída é JSON em std-out, para agente ler, inclusive help e erro de argumento; recusa sai com ok: false e, pelo script trama-agente, status 1.

Usage

```
tr_cli(args = commandArgs(TRUE))
```

Arguments

args Vetor de argumentos, como commandArgs(TRUE).

Details

```
trama-agente state                                    # fluxo na tela: nós, status, arestas
trama-agente catalog [tipo] [--busca T]   # blocos disponíveis / um bloco / os que casam
trama-agente add <tipo> [--id X] [--from no[:porta]]... [--label L] [param=valor]...
trama-agente link <no[:porta]> <no[:porta]>
trama-agente set <no> param=valor...
trama-agente rm <no>
trama-agente op '<json>'                            # op crua (ver schema do documento)
trama-agente apply <arquivo.json>                # lista de ops como um passo de undo
trama-agente result <no> [--wait 30]            # status, resumo, preview, PNG
trama-agente undo
trama-agente explain <tipo>                        # sem editor: o bloco inteiro
trama-agente validate <arquivo.json>            # sem editor: lê, migra e valida
```

catalog, explain e validate funcionam sem editor aberto, sobre as coleções instaladas (ou --colecoes a,b). catalog só faz isso quando não acha editor ou com --offline.

Toda edição (add, link, set, rm, op, apply, undo) aceita --wait N: espera até N segundos o fluxo parar de rodar e devolve, em efeito, o status dos nós tocados e de tudo abaixo deles. Nó blocked traz causa, os ancestrais que falharam.

valor é lido como JSON quando dá (n=3, x=true, v=["a", "b"]) e como texto quando não (coluna=peso). --projeto DIR escolhe qual editor quando há mais de um.

Value

A resposta do editor (lista), invisível; imprime o JSON.

tr_collection	<i>Agrupar tipos, nós, adaptadores e categorias de um domínio.</i>
---------------	--

Description

Uma coleção é o que um pacote de domínio exporta. `js` aponta pra um módulo ES em `inst/` que registra renderers e widgets no front — é o que permite visualização nova sem tocar no núcleo.

Usage

```
tr_collection(
  id,
  version = "0.0.0",
  label = id,
  types = list(),
  nodes = list(),
  adapters = list(),
  categories = list(),
  js = NULL,
  css = NULL,
  transitions = NULL,
  migrations = list(),
  datasets = list()
)
```

Arguments

<code>js</code>	Caminho, relativo ao <code>inst/</code> do pacote, do módulo ES da coleção. Opcional. Os arquivos vizinhos no mesmo diretório também são servidos, pro módulo poder importá-los por caminho relativo.
<code>css</code>	Caminho, relativo ao <code>inst/</code> do pacote, de uma folha de estilo da coleção (as classes que os renderers dela usam). Opcional. Entra na página DEPOIS do <code>trama.css</code> do núcleo, então pode refinar regra do núcleo com a mesma especificidade — e é por isso mesmo que deve prefixar as próprias classes, pra não refinar sem querer.
<code>transitions</code>	<code>data.frame(from, to, n)</code> com quantas vezes um bloco <code>from</code> foi seguido de <code>to</code> nos fluxos de exemplo, ou <code>NULL</code> . É o corpus do sugestor. O <code>to</code> tem que ser da própria coleção: quem é sugerido é o dono do dado, e assim <code>models</code> pode declarar <code>data/ler -> models/ajuste</code> sem o núcleo saber o que é ler nem ajustar. Ver tr_transitions_read() .
<code>migrations</code>	Renomes que a coleção declara pra documentos antigos abrirem já migrados, sem nó órfão. Lista com até três entradas, todas opcionais: <code>nodes</code>: <code>id antigo -> id novo</code> (<code>list("multi/roc" = "models/roc")</code>). O <code>id antigo</code> pode ser de OUTRA coleção — é assim que um bloco muda de casa —, mas o destino tem que ser desta. Quando o nó novo precisa de um param pra se comportar como o antigo, o destino pode ser <code>list(to =</code>

"novo", params = list(nome = valor)): os params entram SÓ nos nós cujo tipo era o id antigo, antes dos renomes de params, e não sobrescrevem param já presente. Em cadeia (a -> b -> c) os params de cada salto se acumulam na ordem. Ex.: "multi/logistic_coefficients" = list(to = "models/coefficients", params = list(exponenciar = TRUE)).

- params: id NOVO do nó -> lista de renomes antigo = list(to = "novo", value = function(v) ..., when = function(v) ...). value é opcional (identidade); when é um predicado sobre o valor antigo, e value só roda quando ele é verdadeiro. Dois casos especiais:
 - converter no lugar: to igual ao nome antigo. Aí when é obrigatório — é ele que reconhece o valor ainda no formato velho e deixa intocado o documento já migrado: confianca = list(to = "confianca", when = is.character, value = function(v) as.numeric(sub("%", "", v)) / 100).
 - derivar vários params: value devolve lista NOMEADA, fundida aos params do nó (param já presente com esse nome vence): intervalo = list(to = "intervalo", when = function(v) grepl("^IC ", v), value = function(v) list(intervalo = "IC", confianca = 0.9)).
 - remover: list(drop = TRUE) (sem to nem value; when opcional) apaga o param que o nó novo não tem mais — idempotente, e o documento regravado simplesmente não o traz: tarefa = list(drop = TRUE).
- ports: id NOVO do nó -> lista antiga = "nova", valendo pra entradas e saídas.

Aplicadas por tr_doc_migrate() quando um fluxo é aberto.

datasets

Bases públicas que a coleção oferece no catálogo de bases, cada uma um [tr_dataset\(\)](#). Só metadados: o dado fica no pacote que o publica e é baixado quando escolhido.

Details

label é o nome que a coleção mostra na aba da paleta; sem ele, a aba escreve o id, que serve mas não explica.

tr_compatible

Compatibilidade de porta — CONSULTADA NUM LUGAR SÓ.

Description

Hoje: igualdade, ou existe adaptador registrado. O insumo espalhava identical(a\$type, b\$type) por três lugares (validação de aresta, isValidConnection no front, inserir-sobre-aresta), o que tornava qualquer evolução de tipagem uma refatoração em três frentes. Sendo função, crescer depois (subtipos, list<T>) custa mudar aqui.

Usage

```
tr_compatible(from_type, to_type, registry = .tr_default_registry)
```

`tr_dataset`*Base de dados pública que uma coleção oferece no catálogo de bases.*

Description

O catálogo guarda só METADADOS: o dado mora no pacote R que o publica e só é baixado quando alguém escolhe a base. Por isso `n` e `variaveis` são declarados à mão, e não lidos: com o pacote ainda não instalado não há de onde lê-los, e é justamente antes de instalar que o usuário quer saber se a base serve.

Usage

```
tr_dataset(  
  pacote,  
  nome,  
  titulo,  
  node,  
  params = list(),  
  descricao = NULL,  
  fonte = NULL,  
  temas = character(),  
  n = NULL,  
  variaveis = NULL,  
  licenca = NULL,  
  url = NULL  
)
```

Arguments

<code>pacote</code>	Pacote R que publica a base.
<code>nome</code>	Nome do objeto no pacote.
<code>titulo</code>	Título curto, para a lista.
<code>node</code>	Id do bloco que carrega a base.
<code>params</code>	Params do bloco (lista nomeada).
<code>descricao</code>	Uma ou duas frases: o que é e para que serve.
<code>fonte</code>	Livro, artigo ou instituição de origem.
<code>temas</code>	Palavras para filtrar (ex.: "regressão", "agricultura").
<code>n, variaveis</code>	Linhas e colunas (número), declarados.
<code>licenca</code>	Licença do dado, como o pacote a declara.
<code>url</code>	Página da base ou do pacote.

Details

O núcleo não sabe carregar dado nenhum. Quem sabe é o bloco node, com os params dados: escolher a base no modal insere esse bloco no canvas, e a prévia e o CSV rodam a função dele. Assim uma base de datasets, de agridat ou de uma API futura passa pelo mesmo caminho, e o fluxo salvo guarda a referência, não o dado.

tr_dataset_load	<i>Carrega uma base rodando o bloco que a declara.</i>
-----------------	--

Description

Só para bloco sem entrada: é isso que o modal promete (a base vira a ORIGEM de um fluxo). Um bloco com várias saídas entrega a primeira.

Usage

```
tr_dataset_load(id, registry = .tr_default_registry)
```

tr_datasets	<i>Bases de todas as coleções do registro, com o estado de instalação.</i>
-------------	--

Description

A mesma base declarada por duas coleções aparece uma vez (a primeira). instalado é lido agora, e não no registro: o usuário instala pacote com o editor aberto, e o catálogo reaberto tem que mostrar isso.

Usage

```
tr_datasets(registry = .tr_default_registry)
```

tr_dependency_collection

JS e CSS de uma coleção — o mecanismo que permite visualização nova sem tocar no núcleo. O JS carrega DEPOIS do runtime e ANTES do editor; o CSS, depois do trama.css do núcleo (a ordem vem de tr_ui()).

Description

Devolve SEMPRE uma lista de dependências, possivelmente vazia. Uma htmlDependency serve UM diretório, então js e css em pastas diferentes precisam de duas — e com nomes distintos, porque o htmltools deduplica dependência por nome e ficaria só com uma delas, sem aviso. No mesmo diretório (o caso comum, inst/trama/) vira uma só, com os dois.

Usage

```
tr_dependency_collection(registry, id)
```

Details

Asset cujo diretório não existe no pacote é pulado, cada um por conta própria — o mesmo silêncio de antes pro js: coleção de globalenv (sem pacote) ou instalada sem o inst/ não derruba o app.

tr_dependency_core *Dependência Shiny com os assets do núcleo (CSS do core e do xyflow, runtime.js, editor.js).*

Description

Dependência Shiny com os assets do núcleo (CSS do core e do xyflow, runtime.js, editor.js).

Usage

```
tr_dependency_core(ver = as.character(utils::packageVersion("trama")))
```

tr_dependency_importmap

Dependência Shiny que injeta o importmap e a captura de erro de boot, antes de qualquer módulo.

Description

Dependência Shiny que injeta o importmap e a captura de erro de boot, antes de qualquer módulo.

Usage

```
tr_dependency_importmap(base)
```

tr_doc	<i>Documento: o grafo que o usuário montou.</i>
--------	---

Description

Semântica de valor — toda op devolve um documento novo, nunca muta. É o que torna undo/redo um log de ops e o que permite comparar duas revisões.

Usage

tr_doc()

Details

Duas camadas no mesmo arquivo, deliberadamente:

- semântica (nodes, edges, seed) entra na chave de cache;
- apresentação (ui) nunca entra. Se posição e parâmetro morassem no mesmo lugar, todo caminho de invalidação precisaria saber ignorar posição — o insumo pagava esse preço com uma função dedicada a comparar "só o que afeta o cálculo".

tr_doc_apply	<i>Conveniência (console, testes, replay). A superfície do transporte é tr_submit() — ela é que carrega a proteção de revisão. Chamar isto direto a partir do servidor reintroduziria a corrida do insumo.</i>
--------------	--

Description

Conveniência (console, testes, replay). A superfície do transporte é tr_submit() — ela é que carrega a proteção de revisão. Chamar isto direto a partir do servidor reintroduziria a corrida do insumo.

Usage

tr_doc_apply(doc, op, registry = .tr_default_registry)

tr_doc_json	<i>Documento como JSON canônico — a forma que trafega pro front e vai pro disco.</i>
-------------	--

Description

Documento como JSON canônico — a forma que trafega pro front e vai pro disco.

Usage

tr_doc_json(doc)

tr_doc_migrate	<i>Leva um documento salvo ao formato do registro atual.</i>
----------------	--

Description

Um só ponto de entrada, dois mecanismos, sempre nesta ordem:

Usage

```
tr_doc_migrate(doc, registry = .tr_default_registry)
```

Details

1. **Migrações de coleção** (`tr_collection(migrations = )`): renomes de tipo de nó (inclusive entre coleções, em cadeia), de param (com conversão de valor) e de porta. Dependem só do NOME antigo, não da versão.
2. **Migrações por versão do nó** (`tr_node(migracoes = )`): funções por versão de destino, aplicadas da `type_version` gravada até a da spec.

Nó cujo TIPO foi renomeado no passo 1 não passa pelo passo 2: a `type_version` gravada é a do nó antigo (outro tipo, outra numeração), e aplicá-la ao destino rodaria funções erradas. O renome declarado é quem leva os params ao formato atual do destino; o nó sai com a `type_version` da spec de destino. Nó não renomeado passa pelos dois: o renome de param da coleção vem antes, e as funções por versão tratam param ausente como já migrado.

Idempotente. O resultado leva `attr(, "migrated") = TRUE` quando algo mudou (o editor re-grava no autosave) e `attr(, "migracoes")` com o registro das subidas de versão (o editor e `tr_project_flow()` avisam).

tr_doc_parse	<i>Lê um documento a partir de texto JSON, validando o formato e re-constituindo vetores achatados pelo parse.</i>
--------------	--

Description

Lê um documento a partir de texto JSON, validando o formato e reconstituindo vetores achatados pelo parse.

Usage

```
tr_doc_parse(txt)
```

tr_doc_read	<i>Lê um documento de um arquivo .json.</i>
-------------	---

Description

Lê um documento de um arquivo .json.

Usage

```
tr_doc_read(path)
```

tr_doc_subset	<i>Recorte do documento com os nós/frames/notas de ids e só as arestas internas.</i>
---------------	--

Description

Aresta que sai do grupo é descartada, e não pendurada: no destino ela apontaria pra um nó que não existe. ids = NULL devolve o documento todo.

Usage

```
tr_doc_subset(doc, ids = NULL)
```

tr_doc_terminals	<i>Nós sem aresta de saída — os alvos default de execução.</i>
------------------	--

Description

Nós sem aresta de saída — os alvos default de execução.

Usage

```
tr_doc_terminals(doc)
```

tr_doc_validate	<i>Revalida um documento inteiro contra um registro.</i>
-----------------	--

Description

Existe por causa de documento escrito à mão ou por LLM — que é uma modalidade de autoria que o formato JSON convida. Por isso checa as mesmas invariantes que as ops garantem (ciclo, aridade de porta, input obrigatório desconectado): um documento escrito à mão viola qualquer uma delas trivialmente, e sem checar aqui quem descobre é o executor — o "erro longe da causa" que a checagem na edição existe pra evitar.

Usage

```
tr_doc_validate(doc, registry = .tr_default_registry)
```

Details

Tipo desconhecido NÃO é erro fatal: vira nó órfão, visível na tela, pra que o usuário veja o que falta em vez de receber um documento que não abre.

tr_doc_write	<i>Grava o documento em path como JSON.</i>
--------------	---

Description

Grava num temporário da MESMA pasta e renomeia, como .tr_atomic no store: o autosave roda a cada op, e um crash ou disco cheio no meio do writeLines truncaria o fluxo do usuário — o único artefato que não se recompõe. Falhar deixa o arquivo anterior intacto.

Usage

```
tr_doc_write(doc, path)
```

tr_errors	<i>Classes de erro do trama.</i>
-----------	----------------------------------

Description

Todo erro do núcleo é classificado (`rlang::abort(class = "tr_error_*")`) e viaja com mensagem acionável — é o que permite ao front escolher o que mostrar e a um modelo corrigir o próprio documento sem ler código. Esta tabela é a referência; um teste garante que toda classe usada no código está aqui, então ela não envelhece em silêncio.

Usage

```
tr_errors()
```

Value

data.frame com class e when.

tr_executor_pool	<i>Executor com pool de daemons (mirai).</i>
------------------	--

Description

Cancelar = MATAR o daemon. Cancelamento cooperativo em R (e no C++ de um terra da vida) não existe; matar é o único contrato honesto. Isso só é aceitável porque o worker é sem estado: o que ele produziu de útil já está no store sob a chave certa, e o que não terminou não deixa rastro (escrita atômica, store.R). Projete o nó para o worker morrer a qualquer momento.

Usage

```
tr_executor_pool(n = 2L, registry = .tr_default_registry, setup = NULL, ...)
```

tr_executor_sequential	<i>Executores: a mesma interface, duas estratégias.</i>
------------------------	---

Description

O executor é plugável de propósito. O sequencial é o default em teste e headless — rápido, determinístico, sem processo extra e sem ramificação de código entre "com UI" e "de script". O pool existe pra que o processo interativo nunca bloqueie.

Usage

```
tr_executor_sequential()
```

Details

Contrato: `$submit(unit, registry, store, ctx_extra = NULL)` -> token `$collect(token)` -> NULL enquanto não terminou, senão `list(ok=, handles=error=) $capacity()` -> quantas unidades em voo ao mesmo tempo `$cancel(tokens)` -> descarta trabalho em andamento `$shutdown()`

`ctx_extra` são as CONFIGURAÇÕES DO RUN que o worker precisa e que não estão na unidade — hoje a cadência de publicação do parcial e a do checkpoint da região (`worker.R`, `stream-driver.R`). Atravessa como dado, o que o contrato deste arquivo já exige de tudo: é lista de escalares, sem função nem ambiente, e por isso serializa para o daemon como a unidade serializa.

Não confundir com o `control.json` de `stream-control.R`: lá vão COMANDOS VIVOS (o clique de pausa, o arrasto do controle de velocidade enquanto a região roda), que mudam durante o run; aqui vão AJUSTES DO RUN, fixos quando ele começa. Juntá-los num arquivo só faria um comando tempo poder mexer no checkpoint em silêncio, e poria política de durabilidade no mesmo arquivo de um gesto de interface. Executor sequencial: roda no próprio processo, na hora.

`cancel()` é honesto sobre não fazer nada: cancelamento cooperativo em R arbitrário não existe, e sem processo separado não há o que matar. O contrato existe nos dois executores mesmo assim, pra que o protocolo não precise mudar depois — mudar protocolo com o front pronto custa dobrado.

tr_export_code	<i>Exporta um fluxo como código R independente do Trama.</i>
----------------	--

Description

Diferente de `tr_flow_code()`, que gera a DSL usada para reconstruir o documento no editor, esta função gera chamadas diretas às funções R dos nós. O script resultante não cria registro, documento, plano ou store do Trama; ele pode ser aberto e executado como qualquer outro script R, desde que os pacotes das coleções estejam instalados.

Usage

```
tr_export_code(
  doc,
  registry = .tr_default_registry,
  format = c("r", "quarto"),
  title = "Análise",
  settings = NULL
)
```

Arguments

doc	Documento ou <code>tr_flow()</code> a exportar.
registry	Registro que resolve os nós e adaptadores do documento.

format	"r" para um script R ou "quarto" para um documento Quarto.
title	Título usado no documento Quarto.
settings	Configurações do projeto, para resolver params de tema como o executor resolve.

Details

Cada nó vira uma variável com o nome do card (`correlograma_acf`, e não o `id`), e um nó de várias saídas é UMA variável: quem a consome lê `ajuste$out`. Os params passados são os efetivos, os mesmos que o executor passa, omitidos só quando coincidem com o default literal da própria função: o script calcula o que o canvas calculou, sem repetir o óbvio.

No formato Quarto, cada card é um chunk; os frames viram seções, na ordem de slide, e as notas viram texto no lugar em que estão. As saídas que nenhum fio consome são as que o fluxo produziu para ser LIDAS, e o documento as mostra (pelo `report` do tipo, quando há; senão, o `print`).

Fluxos ponto a ponto ainda dependem do executor do Trama e, por isso, são recusados em vez de gerar um script que calcularia outra coisa.

Value

Uma string única pronta para ser gravada em um arquivo `.R` ou `.qmd`.

tr_flow	<i>Superfície R para escrever um fluxo — açúcar sobre as ops, nada mais.</i>
---------	--

Description

O documento sempre pôde ser construído no console aplicando listas de ops, que é ótimo para máquina e ruim para gente. O manifesto mira "referência de como se constrói fluxo lógico em R": isso exige que o fluxo se ESCREVA em R idiomático, com pipe, e que o mesmo texto produza o mesmo JSON que a interface. Esta camada não tem semântica própria — cada verbo vira uma op e passa pela mesma validação — para que não existam dois caminhos que possam divergir.

Usage

```
tr_flow(registry = .tr_default_registry, doc = tr_doc())
```

Details

Referência a porta: `"no:porta"`. Os dois pontos são proibidos em `id` de nó (`.tr_check_node_id`), então a separação é inequívoca; `"."` não serviria, porque `id` pode conter ponto.

tr_flow_code	<i>Gera o código R (DSL) que reconstrói o documento. É a outra metade da promessa "diagrama e código são a mesma coisa": o que se monta na interface se lê como R, e o que se escreve em R aparece na interface.</i>
--------------	--

Description

Ordem topológica; params só quando diferem do default; from = quando a ligação é a canônica (primeira saída -> primeira entrada obrigatória livre), senão tr_link() explícito. Posições NÃO entram: são apresentação.

Usage

```
tr_flow_code(doc, registry = .tr_default_registry, registry_expr = "reg")
```

Details

Param cujo NOME é formal de tr_add() (from, label, seed, position, e também type, id, flow) não pode sair dentro do tr_add(): o R casa o nome com o formal, não com o param — é a armadilha que .tr_check_param_shadow() denuncia do lado de quem escreve. Emitido ingenuamente, data/rename (param from) saía como tr_add("ren", "data/rename", from = "mpg", to = "consumo", from = "ler"): argumento duplicado, código que não roda. Esses params saem num tr_set() encadeado — o caminho que o próprio guard manda usar — e, quando o param sombreado é from, as arestas do nó saem por tr_link() explícito, porque o formal from continua vedado para aquele tr_add().

tr_flow_doc	<i>O tr_doc por trás de um tr_flow — a saída pronta pra tr_run(), tr_doc_write() etc.</i>
-------------	---

Description

O tr_doc por trás de um tr_flow — a saída pronta pra tr_run(), tr_doc_write() etc.

Usage

```
tr_flow_doc(flow)
```

tr_fn*Nível 1 da API: a função do nó, chamável direto.*

Description

O insumo mantinha essa propriedade (com teste de equivalência nível 1 vs. grafo) e ela é o que impede a ferramenta de virar gaiola — qualquer nó é inspecionável no console, sem documento e sem cache.

Usage

```
tr_fn(id, registry = .tr_default_registry)
```

tr_galeria_captura*Grava o PNG de uma captura de navegador como item da galeria.*

Description

O nó tem que estar na galeria como captura. png é o PNG em base64, cru ou como data URL (data:image/png;base64,...). Substitui o arquivo anterior do nó, se o nome mudou, e atualiza o manifesto.

Usage

```
tr_galeria_captura(  
  root,  
  store,  
  registry,  
  doc,  
  settings = NULL,  
  handles = list(),  
  node,  
  key,  
  png  
)
```

Arguments

root	Raiz do projeto; a pasta da galeria é <root>/<galeria\$pasta>.
store	Store do projeto (só a exportação o usa; aqui fica pela forma comum).
registry	Registro que resolve os tipos.
doc	Documento.
settings	Settings do projeto; NULL usa os padrões.
handles	Lista nó -> handle, como tr_store_put() grava.

node	Id do nó.
key	Chave do resultado que a captura representa.
png	PNG em base64 (cru ou data URL).

Value

O item gravado, com pronto = TRUE.

tr_galeria_itens	<i>Itens da galeria, em ordem de fluxo, sem exportar nada.</i>
------------------	--

Description

Ordem: topológica pelas arestas; empate pela posição x do card, depois pelo id (para não depender da ordem de inserção). Cada item traz node, rotulo, arquivo, key, tipo ("imagem" ou "captura") e nome (base do arquivo, sem extensão). Um nó marcado TRUE sem resultado válido vira captura com key = NULL.

Usage

```
tr_galeria_itens(root, store, registry, doc, settings = NULL, handles = list())
```

Arguments

root	Raiz do projeto; a pasta da galeria é <root>/<galeria\$pasta>.
store	Store do projeto (só a exportação o usa; aqui fica pela forma comum).
registry	Registro que resolve os tipos.
doc	Documento.
settings	Settings do projeto; NULL usa os padrões.
handles	Lista nó -> handle, como tr_store_put() grava.

Value

Lista de itens (listas nomeadas).

tr_galeria_sincronizar

Sincroniza a pasta da galeria com o projeto.

Description

Exporta os itens-imagem cuja assinatura (key + formato + dpi) mudou ou que não têm arquivo; remove os arquivos do manifesto que saíram da galeria ou trocaram de nome; mantém os itens-captura que já têm PNG. Erro de exportação de um item fica em erro e não derruba os outros.

Usage

```
tr_galeria_sincronizar(
    root,
    store,
    registry,
    doc,
    settings = NULL,
    handles = list()
)
```

Arguments

root	Raiz do projeto; a pasta da galeria é <root>/<galeria\$pasta>.
store	Store do projeto (só a exportação o usa; aqui fica pela forma comum).
registry	Registro que resolve os tipos.
doc	Documento.
settings	Settings do projeto; NULL usa os padrões.
handles	Lista nó -> handle, como tr_store_put() grava.

Value

Itens (como em [tr_galeria_itens\(\)](#)) com versao (assinatura) e pronto (o arquivo existe).

tr_get_node	<i>Busca um tr_node pelo id qualificado (colecão/nome); erro alto se não registrado.</i>
-------------	--

Description

Busca um tr_node pelo id qualificado (colecão/nome); erro alto se não registrado.

Usage

```
tr_get_node(id, registry = .tr_default_registry)
```

tr_get_type	<i>Busca um tr_type pelo id qualificado; erro alto se não registrado.</i>
-------------	---

Description

Busca um tr_type pelo id qualificado; erro alto se não registrado.

Usage

```
tr_get_type(id, registry = .tr_default_registry)
```

tr_handle_failed	<i>Um handle representa erro (não valor) quando carrega \$error.</i>
------------------	--

Description

Um handle representa erro (não valor) quando carrega \$error.

Usage

```
tr_handle_failed(handle)
```

tr_help_test_card	<i>A seção de ajuda que explica o card de teste de hipótese.</i>
-------------------	--

Description

O card trama/test é do núcleo e desenhado igual em toda coleção: régua do p-valor com estrelas quando há p-valor, três pontinhos de nível quando o teste só tem tabela de valores críticos. A explicação mora aqui, escrita UMA vez, pelo mesmo motivo do componente: quinze páginas com quinze cópias divergiam na primeira mudança do renderer (inst/www/runtime.js, VereditoTeste; regas em inst/www/teste.js).

Usage

```
tr_help_test_card()
```

Details

Os cortes vão por extenso ("1 em mil"), e não "0,001": a coleção series confere que todo decimal escrito na ajuda de um teste existe como literal no código do bloco, e o corte do CARD não é do bloco.

Uma coleção que emite um tipo de teste registra o id dele apontando para o renderer do núcleo (registerRenderer("x/test", getRenderer("trama/test"))) e cola esta seção na ajuda de cada bloco de teste.

Value

string markdown, começando por um título de terceiro nível.

tr_icon	<i>Declara o ícone de um bloco: um nome do conjunto Lucide, ou um SVG próprio.</i>
---------	--

Description

Exatamente um dos dois. name é conferido contra os ícones que o sprite realmente tem — sem isso, tr_icon("filter") (que não existe; o nome é list-filter) viraria um buraco silencioso no canto do card, descoberto meses depois. Mesma régua de tr_node() sem description: erro alto, na declaração.

Usage

```
tr_icon(name = NULL, svg = NULL)
```

Details

svg é o MIOLO de um <svg>, não um documento: o elemento e o viewBox de 24×24 são do front, a coleção entrega a geometria. Não é barreira de confiança — uma coleção é um pacote R instalado e já roda JS próprio — é só o que faz o ícone de terceiro herdar cor e tamanho como os outros.

tr_is_template	<i>O texto é um template do trama? Nunca falha: JSON inválido é só "não".</i>
----------------	---

Description

O texto é um template do trama? Nunca falha: JSON inválido é só "não".

Usage

```
tr_is_template(txt)
```

tr_link	<i>Liga from a to. Porta omitida = primeira saída / primeira entrada.</i>
---------	---

Description

Liga from a to. Porta omitida = primeira saída / primeira entrada.

Usage

```
tr_link(flow, from, to, index = NULL)
```

tr_mcp	<i>Servidor MCP do canal de controle.</i>
--------	---

Description

Expõe os verbos do trama-agente como tools do Model Context Protocol, por stdio (JSON-RPC 2.0, uma mensagem por linha). Serve qualquer cliente MCP local: Claude Code, Claude Desktop, Codex, Cursor. Registro de uma linha, por exemplo:

Usage

```
tr_mcp(projeto = ".", entrada = file("stdin"), saida = stdout())
```

Arguments

- projeto Pasta do projeto trama. Sem trama.json, o projeto é criado com todas as coleções trama.* instaladas.
- entrada, saida Conexões do protocolo (o padrão é stdio).

Details

claude mcp add trama -- Rscript -e "trama::tr_mcp()"

Cada tool é um comando do CLI (R/cli.R) montado a partir dos argumentos da tool: não há segundo caminho de edição. As tools que mexem no fluxo precisam do editor aberto; se não houver um para o projeto, o servidor sobe tr_app() num processo R em segundo plano, abre o navegador e espera a aba conectar. O editor que o servidor subiu é encerrado junto com ele; o que a pessoa abriu fica.

stdout é do protocolo. Tudo o que um bloco ou pacote imprimir durante uma tool é desviado para stderr, que o cliente trata como log.

Value

Invisível, quando a entrada fecha.

tr_node	<i>Declara um node type.</i>
---------	------------------------------

Description

Sem efeito colateral: devolve uma especificação. Quem registra é tr_collection() + tr_use(). No insumo, wm_register_node() escrevia no registro E criava o wrapper de nível 1 em globalenv() no ato — o que impedia o projeto de virar pacote e impedia dois registros isolados coexistirem (todo teste compartilhava o mesmo estado global).

Usage

```
tr_node(
  id,
  fn,
  version = 1L,
  label = NULL,
  description,
  help = NULL,
  category = NULL,
  inputs = list(),
  outputs = list(),
  params = list(),
  pure = TRUE,
  fingerprint = NULL,
  volatile = FALSE,
  stochastic = FALSE,
  icon = NULL,
  init = NULL,
  step = NULL,
  role = NULL,
  pressupostos = list(),
  referencias = list(),
  migracoes = list(),
  galeria = FALSE
)
```

Details

`fn` é uma função R comum, usável direto no console — o nível 1 da API. Os argumentos permitidos são os nomes de `inputs`, os de `params`, e os dois reservados `.seed` e `.ctx` (com ponto, pra nunca colidirem com um param que por acaso se chame `seed`).

`description` é obrigatória e `help` é opcional: a primeira é a linha que aparece na paleta e no tooltip do card, a segunda é a página que abre no painel lateral. Obrigar a primeira é o que impede uma coleção de nascer muda — e o catálogo é lido por máquina também, então a ausência custa duas vezes.

`pure = FALSE` exige `fingerprint(params)`: o que do mundo externo entra na chave de cache (mtime+tamanho de arquivo, ETag de URL). Sem isso, um nó que lê um CSV serviria dado velho em silêncio quando o arquivo muda, porque nada no hash teria mudado. `volatile = TRUE` nunca cacheia entre execuções.

`init/step` declaram o nó COM MEMÓRIA: `init(<params>)` devolve o estado inicial, e `step(state, <inputs/params>)` devolve `list(state = , out = )` — sempre essa forma, sem atalho, porque duas formas de retorno obrigariam o driver, o preview e os testes a tratar as duas.

`role` é o papel do bloco no fluxo (ver [tr_category\(\)](#)). Sem ele, vale o da categoria; declare só quando o bloco faz outra coisa que os vizinhos de paleta, como um "Prever" (leitura) dentro de "Modelar" (ajuste).

`pressupostos` e `referencias` são listas de [tr_pressuposto\(\)](#) e [tr_ref\(\)](#): campos genéricos (o núcleo não sabe o que é normalidade), que o catálogo publica para a ajuda do editor e o site

mostrarem igual.

migracoes leva flows salvos com params antigos à versão atual: uma função `function(params)` params por versão de DESTINO, nomeada pelo número — `list(3 = function(p) { p$confiacao <- p$nivel; p$nivel` Ver [tr_doc_migrate\(\)](#).

`galeria = TRUE` diz que o bloco é de visualização: o gráfico dele entra sozinho na galeria do projeto (com `galeria.imagens` ligado em `trama.json`). Os demais blocos, mesmo com preview de imagem, só entram marcados pelo usuário.

tr_node_raiox

Raio-x de um bloco: onde a conta de fato acontece.

Description

Percorre o código da função do bloco e das funções internas do mesmo pacote que ela chama, e devolve as chamadas a funções de OUTROS pacotes (`stats::aov(...)`, `lme4::lmer(...)`), com o trecho de código e o arquivo e a linha em que aparecem. É o que a ajuda e o site mostram em "Implementação": a chamada que faz a conta, e não o invólucro do Trama.

Usage

```
tr_node_raiox(node, ignorar = .tr_raiox_ignorar)
```

Arguments

node	Nó, como devolvido por tr_node() .
ignorar	Pacotes cujas chamadas não entram no raio-x.

Details

Pacotes de infraestrutura (`base`, `rlang`, `tibble`, `cli`...) ficam de fora por padrão: dizem como a saída é montada, não como a conta é feita.

Value

Lista com `funcao` ("pacote::nome"), `arquivo`, `linha` e `chamadas`: uma lista de `list(pacote, funcao, codigo, arquivo, linha, dentro)`, sem repetir o mesmo `pacote::funcao` no mesmo lugar.

tr_op_semantic	<i>Uma op é SEMÂNTICA se não for puramente de apresentação — só ops semânticas disparam re-execução. Um batch é semântico se qualquer op dentro dele for.</i>
----------------	---

Description

Uma op é SEMÂNTICA se não for puramente de apresentação — só ops semânticas disparam re-execução. Um batch é semântico se qualquer op dentro dele for.

Usage

```
tr_op_semantic(op)
```

tr_param	<i>Declara um parâmetro de nó.</i>
----------	------------------------------------

Description

kind decide o widget no front — e o front resolve o widget por um registro (registerWidget), então uma coleção pode trazer kind novo sem o núcleo saber o que ele significa. Os kinds embutidos abaixo existem porque são universais, não porque o núcleo os privilegia.

Usage

```
tr_param(kind, default, label = NULL, ...)
```

Details

default é obrigatório: sem ele o documento teria que carregar valor de todo param de todo nó pra ser interpretável, e um nó recém-criado não teria estado válido.

tr_param_bool	<i>Param booleano.</i>
---------------	------------------------

Description

Param booleano.

Usage

```
tr_param_bool(default = FALSE, label = NULL)
```

tr_param_col

*Param de coluna(s) da tabela que chega por uma entrada.***Description**

role diz que tipo de coluna serve; multi = FALSE é uma coluna só (select); from nomeia a entrada (padrão: a primeira do nó). Continua kind = "cols": o valor é texto ("a" ou "a, b"), igual aos cols já existentes — quem já declara tr_param("cols", ...) segue valendo, só sem as anotações que deixam o front oferecer as colunas do schema da entrada.

Usage

```
tr_param_col(
  default = "",
  label = NULL,
  role = "qualquer",
  multi = FALSE,
  from = NULL,
  example = NULL,
  suggest = TRUE
)
```

Arguments

default	Valor inicial (texto).
label	Rótulo do campo.
role	"numerica", "categorica", "tempo" ou "qualquer".
multi	TRUE aceita várias colunas separadas por vírgula.
from	Nome da entrada cujas colunas servem; NULL é a primeira.
example	Exemplo mostrado no campo.
suggest	FALSE tira o param do preenchimento automático ao conectar (o select ainda oferece "sugerir" a pedido). Para params opcionais: preenchê-los sozinho mudaria a análise sem o usuário pedir.

Details

role inválido aborta com tr_error_bad_param e não com o erro cru de match.arg: é erro de declaração da coleção, da mesma família dos outros que tr_node() levanta.

tr_param_enum	<i>Param de escolha única entre choices.</i>
---------------	--

Description

Param de escolha única entre choices.

Usage

```
tr_param_enum(default, choices, label = NULL)
```

tr_param_int	<i>Param inteiro.</i>
--------------	-----------------------

Description

botao troca o campo por um botão com esse texto, que soma 1 ao valor a cada clique: é o gesto "faça de novo" (baixar de novo, sortear de novo) expresso como param, então entra no hash, no undo e no documento como qualquer outro, e o bloco decide o que fazer quando o número sobe.

Usage

```
tr_param_int(  
  default,  
  min = NA,  
  max = NA,  
  label = NULL,  
  botao = NULL,  
  vazio = NULL,  
  example = NULL  
)
```

Arguments

vazio	Valor que o campo vazio representa, ou NULL.
example	Texto em cinza no campo vazio.

Details

vazio é o valor que o campo VAZIO representa (tipicamente o "automático", como 0 defasagens): com ele igual ao valor, o campo aparece vazio, com example em cinza dizendo o que o vazio quer dizer, e apagar o número volta a ele em vez de acusar "obrigatório".

tr_param_num	<i>Param numérico (ponto flutuante).</i>
--------------	--

Description

Param numérico (ponto flutuante).

Usage

```
tr_param_num(
  default,
  min = NA,
  max = NA,
  step = NULL,
  label = NULL,
  unit = NA_character_
)
```

tr_param_text	<i>Param de texto livre.</i>
---------------	------------------------------

Description

Param de texto livre.

Usage

```
tr_param_text(default = "", label = NULL)
```

tr_param_theme	<i>Param de tema de gráfico.</i>
----------------	----------------------------------

Description

O valor é "padrão" ou o nome de um tema do projeto; o fn recebe a DEFINIÇÃO resolvida (ver `.tr_theme_resolve`), nunca o nome — é a definição que entra na chave de cache, então editar um tema invalida só quem o usa.

Usage

```
tr_param_theme(default = "padrão", label = "Tema")
```

tr_plan	<i>Plano de execução: fecho transitivo dos alvos, em ordem topológica, com a chave de conteúdo de cada unidade já resolvida.</i>
---------	--

Description

Pull-based, herdado do insumo: nó fora do caminho do alvo não executa. O que muda é que aqui o plano é um VALOR inspecionável — dá pra perguntar "o que vai rodar e o que vem do cache" sem executar nada, comparar dois planos pra ver o que uma edição afetou, e decidir cache hit no coordenador.

Usage

```
tr_plan(  
  doc,  
  targets = NULL,  
  registry = .tr_default_registry,  
  store = NULL,  
  settings = NULL  
)
```

Details

A unidade não carrega o tr_node vivo. Carregar carregaria junto o fn e o ambiente dele: numa coleção que mora em globalenv() (console, testes — o "nível 1 da API" que E1 preserva de propósito), despachar isso pra um daemon arrastaria o ambiente inteiro por unidade. O worker recebe node_type e resolve no próprio registro. Efeito colateral bom: o plano vira serializável, coerente com "o plano é um valor".

settings (de project\$settings) resolve os params de tema ANTES do hash. Quem tem projeto em mãos TEM que passá-lo: plano sem ele calcula as chaves com os temas embutidos, e o GC apagaria o cache vivo de quem usa tema do projeto. NULL é a sessão sem projeto.

tr_plan_blocked	<i>Unidades que não podem rodar: falharam, dependem de quem falhou, ou têm porta obrigatória solta.</i>
-----------------	---

Description

Unidades que não podem rodar: falharam, dependem de quem falhou, ou têm porta obrigatória solta.

Usage

```
tr_plan_blocked(plan)
```

tr_plan_keys	<i>Todas as chaves de artefato alcançáveis — o conjunto keep do tr_store_gc().</i>
--------------	--

Description

Todas as chaves de artefato alcançáveis — o conjunto keep do tr_store_gc().

Usage

```
tr_plan_keys(plan)
```

tr_plan_pending	<i>O que de fato precisa rodar.</i>
-----------------	-------------------------------------

Description

O que de fato precisa rodar.

Usage

```
tr_plan_pending(plan)
```

tr_plugins	<i>Plugins do editor</i>
------------	--------------------------

Description

Plugin é um pacote que acrescenta algo ao editor sem ser coleção: não tem blocos e não entra no trama.json do projeto, vale para a instalação inteira (o menu GitHub do trama.github é o caso de uso). O núcleo não conhece nenhuma pelo nome nem depende do que elas fazem; acha as instaladas pelo DESCRIPTION.

Usage

```
tr_plugins(pacotes = getOption("trama.plugins"))
```

Arguments

pacotes	Pacotes a carregar. O padrão é getOption("trama.plugins") e, sem ela, todo pacote trama.* instalado ou carregado que se declara plugin. options(trama.plugins = character()) desliga todos.
---------	---

Details

Contrato: um pacote trama.* com Config/trama/plugin: true no DESCRIPTION e uma função exportada trama_plugin() que devolve uma lista com

- js, css: caminhos dentro de inst/ (como os de coleção), carregados depois das coleções e antes do editor. O JS é módulo e importa de "trama" (por exemplo registerToolBarItem);
- server: function(input, output, session, ctx), chamada uma vez por sessão do editor. ctx\$project() devolve o projeto aberto e ctx\$recarregar() relê do disco o projeto aberto (fluxo, temas, execução), como ao abri-lo; o histórico de desfazer recomeça.

Plugin que falha ao carregar é pulada com aviso: o editor sobe sem ela.

Value

Lista nomeada pelo pacote, com js, css, server, package e version.

tr_port	<i>Declara uma porta.</i>
---------	---------------------------

Description

multiple = TRUE (porta variádica) existe desde o início porque afeta o schema da aresta (index): acrescentar depois seria mudança de formato de documento, não de API. "Combinar N entradas" aparece em todo domínio.

Usage

```
tr_port(type, required = TRUE, multiple = FALSE, stream = FALSE)
```

Details

stream = TRUE marca porta de fluxo: o dado chega ponto a ponto. Ao contrário de multiple, não guarda nada por aresta — logo não toca no schema do documento.

tr_port_default	<i>Porta padrão do editor: 8726, que é "TRAM" no teclado do telefone.</i>
-----------------	---

Description

Fora das faixas que costumam dar conflito — abaixo dos ports efêmeros do Linux (32768–60999, sorteados pelo kernel) e longe dos padrões de ferramenta de dev (3000, 3838, 5173, 8000, 8080, 8787).

Usage

```
tr_port_default()
```

tr_port_free	<i>Primeira porta livre a partir de port.</i>
--------------	---

Description

O teste é abrir um socket servidor e fechar: é o mesmo recurso que o Shiny vai pedir, então não há falso negativo por protocolo. Sobra a corrida entre o fechamento aqui e o bind do Shiny — improvável em uso interativo, e o preço de não manter o socket aberto enquanto o app sobe.

Usage

```
tr_port_free(port, tentativas = 20L)
```

tr_pressuposto	<i>Pressuposto de um bloco estatístico.</i>
----------------	---

Description

verificar aponta para BLOCOS (ids "colecão/nome"), não para funções R: o que a ajuda oferece é "arraste este bloco para conferir", e só um id de nó vira isso no editor. A existência do bloco não é checada aqui porque a coleção que o verifica pode ser carregada depois (ou nem estar instalada).

Usage

```
tr_pressuposto(texto, verificar = NULL, se_falhar = NULL)
```

Arguments

texto	o pressuposto (texto i18n, ver tr_text()).
verificar	NULL ou vetor de ids de nó que ajudam a conferi-lo.
se_falhar	o que fazer se ele não valer (texto i18n, opcional).

tr_preview	<i>Artefato de preview: o que o worker grava e o front desenha.</i>
------------	---

Description

renderer é o id do componente JS que sabe desenhar isso. data viaja no payload (dado já reduzido — downsample, bins, head); files são caminhos relativos no store, pra quando o dado não cabe (imagem raster). Regra: **arquivo só quando o dado não cabe**.

Usage

```
tr_preview(renderer, data = NULL, files = NULL)
```

tr_progress	<i>Lê o progresso publicado por <code>.ctx\$progress()/ctx\$partial()/ctx\$partial_node()</code> pra uma chave, ou NULL se a unidade não publicou nada (ou já terminou — o scheduler limpa ao concluir).</i>
-------------	--

Description

Lê o progresso publicado por `.ctx$progress()/ctx$partial()/ctx$partial_node()` pra uma chave, ou NULL se a unidade não publicou nada (ou já terminou — o scheduler limpa ao concluir).

Usage

```
tr_progress(store, key)
```

tr_project	<i>Projeto: a pasta que reúne documento, store e manifesto de coleções.</i>
------------	---

Description

Existe porque três coisas no insumo eram caminhos fixos no código — o diretório de blocos, o de blueprints e o que o "usos de bloco" varria. Sem um conceito de projeto, "abri e deu tipo de nó desconhecido" não tem diagnóstico: o manifesto é o que permite dizer QUAL coleção falta.

Usage

```
tr_project(root = ".", collections = character(), create = TRUE)
```

Details

meu-projeto/ trama.json coleções, executor flows/*.json documentos .trama/store/ artefatos (gitignore)

tr_project_at	<i>Abre um projeto REUSANDO um registry já carregado.</i>
---------------	---

Description

É o que permite trocar de projeto com o editor em pé. `tr_project()` monta um registry novo, carregando cada coleção do manifesto — e é isso que não se pode fazer numa página que já está aberta: o JS e o CSS de cada coleção entram como `htmlDependency` quando a UI é montada (`tr_ui()`), uma vez só. Registry novo com coleção nova no R e nenhum renderer no navegador seria um card verde sem desenho nenhum.

Usage

```
tr_project_at(root, registry)
```

Details

Daí a recusa ser a regra, e não um remendo: se o manifesto pede coleção que a página não tem, isto aborta NOMEANDO-A, antes de criar pasta ou tocar em qualquer estado. É o diagnóstico para o qual o manifesto existe.

A comparação é contra o PACOTE (\$package), não contra o id da coleção: o manifesto guarda trama.data e o registry indexa por data, e os dois só coincidem por acidente (ver tr_use()).

tr_project_flow	<i>Lê o flow name do projeto; documento vazio se o arquivo ainda não existe.</i>
-----------------	--

Description

É AQUI que as migrações das coleções entram (tr_doc_migrate()): este é o ponto único em que um documento de disco encontra o registro que vai interpretá-lo — o editor abre por aqui tanto na subida quanto ao trocar de projeto, e importar grava e reabre por aqui. tr_doc_read() não tem registro, e tr_doc_validate() é só diagnóstico (não devolve documento).

Usage

```
tr_project_flow(project, name = "main")
```

tr_project_gc	<i>GC do projeto: preserva tudo que QUALQUER flow do projeto alcança. tr_store_gc recebia keep de um documento só; com dois flows no mesmo projeto, coletar pelo primeiro apagaria o cache do segundo. Falha de leitura de um flow não impede o GC dos outros, mas aborta a coleta — melhor não coletar do que apagar o que um flow ilegível alcançava. Os settings do projeto entram no plano pelo mesmo motivo: sem eles a chave de quem usa tema do projeto seria outra, e o cache vivo pareceria órfão.</i>
---------------	---

Description

GC do projeto: preserva tudo que QUALQUER flow do projeto alcança. tr_store_gc recebia keep de um documento só; com dois flows no mesmo projeto, coletar pelo primeiro apagaria o cache do segundo. Falha de leitura de um flow não impede o GC dos outros, mas aborta a coleta — melhor não coletar do que apagar o que um flow ilegível alcançava. Os settings do projeto entram no plano pelo mesmo motivo: sem eles a chave de quem usa tema do projeto seria outra, e o cache vivo pareceria órfão.

Usage

```
tr_project_gc(project, max_age_days = 7)
```

tr_project_new	<i>Cria um projeto novo: as pastas e o manifesto.</i>
----------------	---

Description

tr_project() Lê o trama.json e nunca o escreve — os manifestos do repositório foram escritos à mão. Fazer create = TRUE passar a gravá-lo seria pior que a lacuna: tr_project() é chamado sobre projeto que JÁ existe o tempo todo, e ele reescreveria o manifesto alheio com as coleções que por acaso lhe passaram, em silêncio. Criar é outro verbo.

Usage

```
tr_project_new(root, collections = character())
```

Details

collections são nomes de PACOTE, como no manifesto. Quem chama pela interface passa as coleções que a página tem — é a única escolha que produz projeto abrível ali mesmo, sem recarregar.

tr_project_save	<i>Grava o flow name do projeto em flows/<name>.json.</i>
-----------------	---

Description

Grava o flow name do projeto em flows/<name>.json.

Usage

```
tr_project_save(project, doc, name = "main")
```

`tr_project_set_galeria`*Grava a configuração da galeria (galeria em trama.json).*

Description

Mesmo molde de `tr_project_set_sugestoes()`: valida antes de abrir o arquivo e mexe só na chave `galeria`. Campo ausente em `galeria` vale o padrão (pasta `gallery`, imagens entram sozinhas, PNG a 300 dpi).

Usage

```
tr_project_set_galeria(root, galeria)
```

Arguments

<code>root</code>	Pasta do projeto (precisa ter <code>trama.json</code>).
<code>galeria</code>	Lista com pasta, imagens, formato e dpi.

Value

Os settings do projeto já com a galeria nova, invisível.

`tr_project_set_marca` *Liga ou desliga a marca d'água dos frames exportados.*

Description

Verbo separado de `tr_project_set_themes()` porque a escolha é separada: o painel salva as duas coisas no mesmo gesto, mas quem chama do console quer desligar a marca sem ter que reescrever a lista de temas inteira para isso.

Usage

```
tr_project_set_marca(root, mostrar)
```

Arguments

<code>root</code>	Pasta do projeto (precisa ter <code>trama.json</code>).
<code>mostrar</code>	TRUE para carimbar o PNG exportado, FALSE para não carimbar. Chama-se <code>mostrar</code> , e não <code>marca</code> , porque é assim que se lê no chamador: <code>tr_project_set_marca(root, mostrar = FALSE)</code> .

Details

Valida ANTES de abrir o arquivo, pelo mesmo motivo dos temas: valor que `.tr_settings()` recusa tranca o projeto fora do editor. Mexe só na chave `marca`; o resto do manifesto volta como estava.

Value

Os settings do projeto já com a `marca` nova, invisível.

`tr_project_set_sugestoes`

Liga ou desliga as sugestões automáticas de params de coluna.

Description

Espelha `tr_project_set_marca()`: chave própria do manifesto, validada antes de abrir o arquivo, e o resto do `trama.json` volta como estava. Desligada, o editor não pré-preenche nada ao conectar; o que já foi sugerido fica.

Usage

```
tr_project_set_sugestoes(root, ligar)
```

Arguments

<code>root</code>	Pasta do projeto (precisa ter <code>trama.json</code>).
<code>ligar</code>	TRUE para sugerir, FALSE para não sugerir.

Value

Os settings do projeto já com a `flag` nova, invisível.

`tr_project_set_themes` *Grava os temas no manifesto — o único verbo que escreve temas.*

Description

Valida ANTES de abrir o arquivo: manifesto com tema quebrado não abre o projeto (`tr_project_at()` recusa), então gravar e validar depois deixaria o usuário trancado fora do próprio projeto.

Usage

```
tr_project_set_themes(root, temas, padrao)
```

Arguments

root	Pasta do projeto (precisa ter trama.json).
temas	Lista nomeada de temas (campos como em trama.json). Lista vazia grava explicitamente os três temas embutidos (escuro, claro, clássico), e não um temas vazio.
padrao	Nome do tema padrão (obrigatório): uma string não vazia que esteja em temas.

Details

paleta vai com I() pelo mesmo motivo de collections em .tr_cfg_rewrite(): uma cor só não pode sair como string onde oito saem como array. padrao é obrigatório aqui, ao contrário da leitura: escolher o padrão calado só serve pra manifesto antigo ou editado à mão. Quem GRAVA (o painel) sabe qual escolheu, e mensagem sem o campo é bug do front — gravar "escuro" ou o primeiro tema no lugar esconderia o bug.

Value

Os settings normalizados, invisível.

tr_provenance	<i>Proveniência dos resultados em cache de um fluxo.</i>
---------------	--

Description

Para cada nó do fluxo, de onde veio o valor que o editor mostra: a chave, quando foi produzido, com que versão de R e de cada pacote de coleção. A chave de cache NÃO inclui versão de dependência (ver hash.R), então um resultado pode ter sido calculado por outra versão da coleção — esta tabela é como se descobre, e é o que um relatório reprodutível precisa citar.

Usage

```
tr_provenance(project, flow = "main")
```

Arguments

project	Projeto (tr_project()).
flow	Nome do fluxo em flows/.

Details

Nós sem resultado em cache saem com status = "pendente"; resultados gravados antes de o handle guardar o ambiente saem com r = NA.

Value

data.frame com node, type, port, status, key, created, r e uma coluna pkg_<pacote> por pacote registrado no handle.

tr_ref	<i>Referência de um bloco.</i>
--------	--------------------------------

Description

O papel diz POR QUE a fonte está ali: a teoria do método, o livro-texto que o ensina, o pacote que o implementa. A de implementação é outra espécie — o autor sai de `utils::citation(pacote)` e a versão é lida do pacote instalado na hora do catálogo —, por isso ela exige pacote e função e dispensa autores, ano e título.

Usage

```
tr_ref(  
  autores = NULL,  
  ano = NULL,  
  titulo = NULL,  
  fonte = NULL,  
  doi = NULL,  
  url = NULL,  
  papel = "teoria",  
  pacote = NULL,  
  funcao = NULL,  
  nota = NULL  
)
```

Details

doi vai sem prefixo de URL para haver UMA forma dele: com e sem `https://doi.org/` convivendo, o front teria que adivinhar qual montar.

tr_registry	<i>Registro: onde tipos, nós e adaptadores vivem depois de carregados.</i>
-------------	--

Description

É um objeto, não estado global. O insumo escrevia em `globalenv()`, o que (a) impede virar pacote e (b) faz todo teste compartilhar o mesmo estado — um registro por teste é o que permite testar coleções em isolamento. O registro default do processo existe só pra conveniência interativa.

Usage

```
tr_registry()
```

tr_registry_for	<i>Reconstrói um registro a partir dos NOMES DE PACOTE — como o worker recupera o catálogo sem receber funções pela fronteira.</i>
-----------------	--

Description

Reconstrói um registro a partir dos NOMES DE PACOTE — como o worker recupera o catálogo sem receber funções pela fronteira.

Usage

```
tr_registry_for(packages)
```

tr_replay	<i>Reaplica um log de ops sobre doc — a base do undo.</i>
-----------	---

Description

Reexecutar o log é mais simples e mais confiável que inverter cada op (disconnect inverso precisaria lembrar o index; remove_node inverso precisaria lembrar params, seed e posição). O documento é pequeno e as ops são baratas — reconstruir é o caminho honesto.

Usage

```
tr_replay(ops, registry = .tr_default_registry, doc = tr_doc())
```

Details

doc tem que ser o documento em que o log COMEÇOU. O default vazio só serve pra um log que cobre a história inteira; o undo do editor não é esse caso — ver .tr_undo_doc().

tr_retry	<i>Comando: limpa SÓ o(s) handle(s) de erro de um nó, preservando o checkpoint.</i>
----------	---

Description

O gesto explícito que faltou depois da Fase 5. Uma falha grava handle de ERRO sob toda porta de saída da unidade (R/scheduler.R, o ramo de falha de collect(), via tr_store_put_error); tr_plan() marca a unidade failed, e o scheduler a serve direto de skipped/from_cache — tr_run() sozinho, no MESMO store, nunca redespacha, mesmo com um ckpt.rds bom esperando no disco (R/stream-driver.R: falha no step, falha no colapso e worker morto guardam o checkpoint DE PROPÓSITO). O único jeito público de tirar o handle de erro era tr_bust() — que apaga <store>/stream/ inteiro (R/hash.R), porque bustar significa "o código mudou sem a chave mudar", e aí retomar serviria um histórico metade velho, metade novo. Usado contra um erro comum, tr_bust() jogaria fora os pontos bons do checkpoint pela mesma chamada que limpa o erro do último ponto — exatamente o custo que a retenção do checkpoint existe pra evitar.

Usage

```
tr_retry(doc, node, registry = .tr_default_registry, store, settings = NULL)
```

Details

`tr_retry()` faz só a metade que falta: tira o(s) handle(s) de erro da unidade e NUNCA toca `<store>/stream/<chave da unidade>/`. Não é `tr_bust()` com um filtro a mais — é o oposto dele: um preserva o checkpoint de propósito, o outro apaga de propósito, e os dois precisam continuar existindo porque servem casos diferentes ("o ponto 10.000 errou, quero retomar dali" vs. "mudei o código, o checkpoint velho não presta mais").

Recusa apagar um handle BOM (`tr_handle_failed()` falso): um artefato válido não é candidato a retry — apagá-lo seria um `tr_bust()` disfarçado, silencioso, sem o comentário que explica a perda do checkpoint. A recusa é TRANSACIONAL: se qualquer porta de saída da unidade guarda um artefato bom, a chamada aborta ANTES de apagar qualquer handle de erro das outras portas — uma região de dois colapsos não pode sair com um retry parcial.

Nó sem handle nenhum (nunca rodou, ou já foi limpo) é NO-OP silencioso: a mesma forma que `tr_stream_command()` escolheu pra chave sem diretório (`R/stream-control.R`) — clique perdido não é erro.

Membro INTERIOR de uma região não tem unidade nem chave de saída própria (ver o comentário de `tr_value()` acima, e `plan$region_of`): pedir retry dele significa "retry a região", resolvida pelo mesmo mapa que `tr_value()` já usa pra leitura — o efeito observável é o de pedir retry por qualquer um dos colapsos dela.

`settings` existe pelo mesmo motivo que em `tr_value()`: sem ele o plano calcularia a chave com o tema embutido, e talvez não achasse a MESMA unidade (nem as MESMAS chaves de saída) que o run de verdade usou.

Não roda nada — só planeja, pra achar a unidade e as chaves, e mexe no store. É COMANDO, não op de documento: não toca doc, rev nem o log de undo (Decisão 10), do mesmo jeito que `pause/step/stop` de `R/stream-control.R`.

Value

`invisible(TRUE)` se limpou pelo menos um handle de erro; `invisible(FALSE)` se não havia nada a limpar (nó sem handle). Vale em nó COMUM também, e não só em região: ali limpa o erro cacheado e o nó roda de novo — só não há de onde retomar, porque nó comum não tem checkpoint. É por isso que a recusa não fala de checkpoint: o `tr_bust()` que ela indica serve para os dois casos, e o que ele apaga a mais (o diretório `stream/`) só existe num deles.

SEM entrada no barramento do Shiny, de propósito: `transport.R` só decodifica comando VIVO cuja chave o front já tem em mão (`pause`, `step`, `stop`). Retentar precisa de `doc + registry + settings` para resolver um id de nó numa unidade, que é conhecimento de plano dentro do barramento — o que aquele arquivo recusa por desenho. Consequência a registrar em vez de descobrir: um botão de retentar no front precisa de mensagem nova ou de um caminho por chave, e isso é trabalho da fase da interface.

tr_run

Coordenador: monta o plano, despacha o que falta, emite eventos.

Description

Pull-based herdado do insumo — nó fora do caminho do alvo não roda. O "acender os cards em sequência" continua saindo de graça da ordem real de execução: o coordenador não calcula fila nenhuma, só emite o que acontece.

Usage

```
tr_run(
  doc,
  targets = NULL,
  registry = .tr_default_registry,
  store,
  executor = tr_executor_sequential(),
  on_event = function(ev) invisible(NULL),
  run_id = NULL,
  settings = NULL,
  ctx_extra = NULL
)
```

Details

on_event(ev) recebe, por unidade: cached, failed, invalid, blocked, running, done, cancelled (unidade superada por um run novo, ou região PARADA por comando); mais progress e partial de quem publica pelo .ctx; e um run_finished no fim. É o contrato que o transporte (E5) traduz pra mensagem de front — o motor não sabe o que é Shiny.

ctx_extra é a PORTA DOS AJUSTES DO RUN: uma lista de escalares que desce inteira até o .ctx do worker (.tr_make_ctx()), pelo scheduler e pelo executor, e vale igual nos dois executores. Hoje é lida por: publish_every — segundos entre publicações de parcial (default 0.1) checkpoint_every — passos entre checkpoints da região (default 100; 0 desliga a serialização periódica do acumulador)

Não é settings, e a distinção é de desenho: settings é conteúdo do projeto e ENTRA no hash (o tema muda o gráfico), enquanto ctx_extra é ajuste de sessão e não toca documento nem chave (Decisão 9) — se tocasse, baixar a cadência de checkpoint recomputaria o fluxo inteiro que ela existe pra proteger. Também não é tr_stream_command(): lá vão comandos VIVOS, que mudam durante o run; aqui vão ajustes fixados quando o run começa.

Existe como argumento explícito, e não como opção global, porque duas sessões sobre o mesmo store são o caso normal (é como os testes encenam o segundo processo) e um options() faria a cadência de uma vaziar na outra.

tr_run_logger	<i>Log de execução: uma linha JSON por evento, um arquivo por run.</i>
---------------	--

Description

O motor já emite duração real e chave por unidade; até aqui isso morria no front. Gravar é o que dá base ao "Monitorar" do manifesto sem inventar superfície nova — e é telemetria honesta para medir o que o cache poupa. Handles ficam de fora: são pesados e já estão no store.

Usage

```
tr_run_logger(root, run_id)
```

tr_run_plan	<i>Executa um plano já montado — separado de tr_run() porque o coordenador interativo (E5) monta o plano a cada edição pra decidir o que cancelar, e só então manda executar.</i>
-------------	---

Description

Executa um plano já montado — separado de tr_run() porque o coordenador interativo (E5) monta o plano a cada edição pra decidir o que cancelar, e só então manda executar.

Usage

```
tr_run_plan(  
  plan,  
  registry = .tr_default_registry,  
  store,  
  executor = tr_executor_sequential(),  
  on_event = function(ev) invisible(NULL),  
  run_id = NULL,  
  ctx_extra = NULL  
)
```

tr_runs	<i>Lê todos os runs de um projeto como data.frame (um evento por linha).</i>
---------	--

Description

Lê todos os runs de um projeto como data.frame (um evento por linha).

Usage

```
tr_runs(project)
```

tr_scheduler

*Scheduler: a execução como máquina de passos.***Description**

tr_run_plan() era um while com Sys.sleep, e tr_server o chamava dentro de um observeEvent — o processo do Shiny ficava travado até o run acabar, MESMO com o pool mirai. Três promessas do design dependiam de o coordenador ficar livre e não existiam: cancelar unidade superada, mostrar progresso enquanto roda, mostrar parcial. Aqui o estado é explícito e step() faz um passo (despacha o que está pronto, coleta o que terminou). Headless: while (!s\$step()) Shiny: later::later(step).

Usage

```
tr_scheduler(
  plan,
  registry = .tr_default_registry,
  store,
  executor = tr_executor_sequential(),
  on_event = function(ev) invisible(NULL),
  run_id = NULL,
  inherit = list(),
  ctx_extra = NULL
)
```

Details

DOIS conjuntos, e a distinção importa: done — terminou com valor bom (executou agora, ou veio do cache). skipped — não vai produzir valor (falhou, inválido, bloqueado).

A versão anterior inferia "terminou" de "não está mais em pending" — e uma unidade EM VOO também não está em pending. Com capacidade > 1 o consumidor era despachado antes do produtor gravar, falhava com "chave ausente", e — pior — esse erro virava handle no store: como a chave é determinística, o erro falso ficava cacheado com a mesma autoridade de um resultado bom, para sempre.

tr_server

*Transporte: UM canal pra cima, UM pra baixo.***Description**

O Shiny é barramento, não lógica. Nenhuma regra de domínio, de documento ou de execução mora aqui — decodifica, chama o motor, codifica de volta.

Usage

```
tr_server(
  project,
  flow = "main",
  executor = tr_executor_sequential(),
  autosave = TRUE,
  ctx_extra = NULL,
  plugins = list()
)
```

Details

Pra cima: `input$str_op` com `{seq, base_rev, op}`. Pra baixo: `sendCustomMessage("tr_event", ...)`.

O front emite OPS, nunca o documento inteiro. No insumo o front empurrava o grafo todo com `debounce` e o servidor `diffava`; havia ainda um segundo timer pedindo execução em paralelo, sem garantia de ordem. Quando a ordem não valia, a engine rodava contra o documento antigo, via que nada mudou, servia do cache e o preview não atualizava — sem erro, em silêncio.

tr_set

*Aplica set_param para cada ... nomeado, um por um.***Description**

Aplica `set_param` para cada ... nomeado, um por um.

Usage

```
tr_set(flow, id, ...)
```

tr_store

*Store endereçado por conteúdo — o protocolo, não um cache.***Description**

É o que permite o worker ser outro processo sem serializar objeto vivo: o worker computa, grava o artefato e o preview sob a chave, e devolve **só o handle**. O coordenador nunca carrega um valor pesado.

Usage

```
tr_store(root, project_root = NULL)
```

Details

Layout: objects/. o valor, gravado pelo store do TIPO handles/.json metadados: tipo, tamanho, resumo, preview, erro previews/. artefato de preview, quando tem arquivo progress/.json progresso e parcial de uma unidade EM VOO stream//ckpt.rds checkpoint de uma região de fluxo em andamento stream//control.json pause/step/tempo da região em voo
progress/ e stream/ são de trabalho em voo, não de artefato: nascem quando a unidade começa e saem quando ela conclui. stream/ é por chave de UNIDADE (não de saída), criado pelo driver quando o laço começa.

Escrita atômica (temp + rename) não é refinamento: o contrato do executor é que o worker pode morrer a qualquer momento (cancelar = matar o daemon, porque cancelamento cooperativo em R não existe). Um arquivo parcial sob uma chave válida seria indistinguível de um resultado bom, para sempre — o cache serviria lixo silenciosamente.

tr_store_gc	<i>Coleta de lixo: o store cresce sem limite por construção (cada edição de param cria uma chave nova). keep são as chaves alcançáveis do documento atual; o resto sai por idade.</i>
-------------	---

Description

Coleta de lixo: o store cresce sem limite por construção (cada edição de param cria uma chave nova). keep são as chaves alcançáveis do documento atual; o resto sai por idade.

Usage

```
tr_store_gc(store, keep = character(), max_age_days = 7)
```

tr_store_get	<i>Lê o valor sob key, restaurado pelo tr_type; erro alto se a chave estiver ausente ou guardar um erro.</i>
--------------	--

Description

Lê o valor sob key, restaurado pelo tr_type; erro alto se a chave estiver ausente ou guardar um erro.

Usage

```
tr_store_get(store, key, type_spec)
```

tr_store_handle	<i>Handle ilegível conta como ausente, nunca como exceção.</i>
-----------------	--

Description

Sem isto, um único JSON truncado (worker morto, disco cheio) abortava `tr_store_gc()` e `tr_bust()` — deixando o store PERMANENTEMENTE incoletável, num sistema que cresce sem limite por construção. Disco cheio esperando acontecer, sem caminho de recuperação dentro da API.

Usage

```
tr_store_handle(store, key)
```

tr_store_has	<i>Uma chave só "existe" se handle E objeto existem. Handle órfão (remoção externa, GC interrompido) faria o plano marcar cached = TRUE e a unidade ser pulada — e o consumidor a jusante explodiria com erro nu de conexão.</i>
--------------	--

Description

Uma chave só "existe" se handle E objeto existem. Handle órfão (remoção externa, GC interrompido) faria o plano marcar `cached = TRUE` e a unidade ser pulada — e o consumidor a jusante explodiria com erro nu de conexão.

Usage

```
tr_store_has(store, key)
```

tr_store_path	<i>Resolve um caminho relativo contra a raiz do projeto. Absoluto passa direto; sem raiz conhecida (store solto em teste), cai no cwd — que é o que acontecia antes, só que agora explícito.</i>
---------------	--

Description

Resolve um caminho relativo contra a raiz do projeto. Absoluto passa direto; sem raiz conhecida (store solto em teste), cai no cwd — que é o que acontecia antes, só que agora explícito.

Usage

```
tr_store_path(store, path)
```

tr_store_put	<i>Grava valor + handle (+ preview). O handle é escrito POR ÚLTIMO: é ele que marca a chave como presente (tr_store_has), então um worker morto no meio deixa no máximo um objeto órfão — nunca um handle apontando pra nada.</i>
--------------	---

Description

collections é o conjunto de coleções que produziram o artefato. Existe porque tr_bust(store, coleção) deriva a coleção de node_type, e isso não funciona quando a unidade não é um nó: a região de fluxo grava sob node_type = "trama/stream_region" — que lê como coleção "trama" — e mistura coleções de verdade. Ausente (todo nó comum), tr_bust() cai no node_type, que é onde a informação sempre esteve.

Usage

```
tr_store_put(
    store,
    key,
    value,
    type_spec,
    node_type = NULL,
    duration = NA_real_,
    collections = NULL,
    env = NULL,
    tema = NULL
)
```

tr_store_put_error	<i>Erro é VALOR, não exceção que some.</i>
--------------------	--

Description

Sem isto, um nó que falha não deixa registro, e o jusante tenta executar a cada edição — reexecutando o mundo por causa de uma ponta quebrada. Com o handle de erro, o plano marca o jusante como bloqueado e não roda nada.

Usage

```
tr_store_put_error(
    store,
    key,
    message,
    class = NULL,
    traceback = NULL,
```

```

    node_type = NULL,
    collections = NULL
)

```

tr_stream_command	<i>Manda um comando para a região que roda sob key (a chave da UNIDADE, que é a que vem nos eventos do run).</i>
-------------------	--

Description

cmd é um de: "play" — volta a andar sozinho "pause" — para entre passos e espera "step" — um passo e volta a pausado "tempo" — muda o atraso por passo, sem mexer em pausado/rodando "stop" — encerra o run desta região deixando checkpoint

Usage

```
tr_stream_command(store, key, cmd, tempo = NULL)
```

Details

Chave sem diretório de região é NO-OP silencioso: o clique do usuário e o evento done se cruzam, e comando de região que já terminou é normal, não erro. Erro ali viraria banner vermelho por um botão que chegou tarde — e criar o diretório "por cortesia" encheria o store de diretórios de chaves que nunca rodaram. O diretório é criado pelo DRIVER quando a região começa: é ele quem sabe que há laço pra comandar.

tr_submit	<i>Protocolo de op: {seq, base_rev, op}.</i>
-----------	--

Description

O front emite OPS, nunca o documento inteiro. No insumo o front empurrava o grafo todo com debounce e o servidor diffava pra descobrir se algo relevante tinha mudado — e havia um segundo timer pedindo execução em paralelo, sem garantia de ordem de chegada. Quando a ordem não valia, a engine rodava contra o documento ANTIGO, via que nada mudou, servia do cache, e o preview não atualizava: sem erro, silenciosamente.

Usage

```
tr_submit(doc, envelope, registry = .tr_default_registry)
```

Details

Aqui essa corrida vira erro explícito: se base_rev não bate com a revisão atual, a op é recusada e o cliente ressincroniza com o documento inteiro. Estado nunca fica corrompido em silêncio.

seq existe por uma armadilha do Shiny: input\$x ignora valores idênticos consecutivos. Duas ops iguais em sequência (clicar "remover" no mesmo nó duas vezes, mover pro mesmo lugar) desapareceriam sem um campo que muda sempre.

tr_template	<i>Envelope de template a partir de um documento.</i>
-------------	---

Description

Só a estrutura viaja: a posição é normalizada pro canto (0,0) e colecoes lista os pacotes que quem cola precisa ter.

Usage

```
tr_template(doc, nome, descricao = "", registry = .tr_default_registry)
```

tr_template_dir	<i>Pasta de templates por escopo.</i>
-----------------	---------------------------------------

Description

biblioteca é pessoal (vale em qualquer projeto); projeto viaja junto com a pasta do projeto. Templates de coleção moram em inst/templates do pacote e não têm pasta gravável: só são listados.

Usage

```
tr_template_dir(escopo = c("biblioteca", "projeto"), root = ".")
```

tr_template_json	<i>Template como JSON (envelope + documento canônico).</i>
------------------	--

Description

Template como JSON (envelope + documento canônico).

Usage

```
tr_template_json(tpl)
```

tr_template_list	<i>Templates disponíveis: coleções carregadas, biblioteca pessoal, projeto.</i>
------------------	---

Description

Uma lista de registros (escopo, nome, descricao, arquivo, colecoes). Arquivo que não é template — dado solto, JSON quebrado — é ignorado: uma pasta suja não pode esconder os templates bons.

Usage

```
tr_template_list(root = ".", registry = .tr_default_registry)
```

tr_template_op	<i>Uma op batch que insere o template com o canto em origin, ids novos.</i>
----------------	---

Description

Ids novos a cada chamada: colar o mesmo template duas vezes não pode colidir. Uma op só (e não várias) pra inserção ser um passo de undo. Frames vêm antes dos nós (ficam por baixo); tamanho, vista, modo e arestas vêm depois, porque exigem o nó já criado.

Usage

```
tr_template_op(tpl, origin = c(0, 0))
```

tr_template_parse	<i>Lê um template a partir de texto JSON.</i>
-------------------	---

Description

Lê um template a partir de texto JSON.

Usage

```
tr_template_parse(txt)
```

tr_template_read	<i>Lê um template de um arquivo .json.</i>
------------------	--

Description

Lê um template de um arquivo .json.

Usage

```
tr_template_read(path)
```

tr_template_save	<i>Grava o template em dir; nome de arquivo derivado do nome do template.</i>
------------------	---

Description

Grava o template em dir; nome de arquivo derivado do nome do template.

Usage

```
tr_template_save(tpl, dir, overwrite = FALSE)
```

tr_test	<i>O registro de um teste de hipótese, na forma que o card trama/test desenha.</i>
---------	--

Description

Precisa de uma fonte de decisão: p_valor ou criticos com o nível "5%". Sem nenhuma das duas o teste viraria um "não rejeita H0" confiante, e isso é erro aqui, não card.

Usage

```
tr_test(
  teste,
  h0,
  estatistica,
  rotulo_estat = "estatística",
  p_valor = NA_real_,
  gl = NA_character_,
  criticos = NULL,
  sentido = "menor",
  conclusao_sim,
```

```

    conclusao_ao,
    efeito = NULL,
    nota = "",
    fonte = "",
    extra = NULL,
    classe = NULL
)

```

Arguments

teste, h0	nome do teste e a hipótese nula, em texto.
estatistica	o valor da estatística; rotulo_estat como ela se escreve.
p_valor	o p-valor, ou NA quando o teste só tem tabela de críticos.
gl	graus de liberdade: texto ("2; 27"), usado como veio, ou número(s), formatados aqui — inteiro sem casas, fracionário (Welch, Satterthwaite) com uma casa e vírgula decimal, vários separados por "; " — ou NA.
criticos	vetor nomeado de valores críticos ("10%", "5%", "1%").
sentido	"menor" ou "maior": a cauda em que a estatística rejeita contra os críticos. Ignorado quando há p-valor.
conclusao_sim, conclusao_ao	a conclusão em uma linha, conforme rejeite ou não H0 a 5%.
efeito	list(rotulo, valor, li, ls) — o tamanho do efeito, com li e ls o intervalo de confiança de 95% — ou NULL. Um campo nivel opcional, se vier, tem de ser 0.95: as colunas da tabela são efeito_li_95/efeito_ls_95, fixas para empilhar, e um IC de 90% nelas mentiria em silêncio.
nota, fonte	texto livre; extra lista nomeada de valores por teste, com quaisquer nomes (na tabela, viram colunas extra_<nome>).
classe	classe S3 a mais, na frente de tr_test (a coleção de origem).

Value

objeto tr_test.

tr_test_table	<i>Teste -> UMA linha de data.frame, com colunas FIXAS.</i>
---------------	--

Description

Sempre as mesmas colunas, com NA onde não se aplica (teste de tabela não tem p-valor; quase nenhum tem efeito): o pesquisador empilha testes de coleções diferentes num quadro só, e esquema estável é o que deixa o empilhamento ser um relatório. Só extra vira coluna quando existe — é, por definição, o que cada teste tem de próprio —, e com prefixo extra_ e o nome como veio: as chaves podem ser nomes de contraste do usuário ("A - B", até "nota"), e sem prefixo colidiriam com as fixas; sem check.names = FALSE, "A - B" viraria "A...B".

Usage

```
tr_test_table(x)
```

Arguments

x um tr_test.

Value

data.frame de uma linha.

tr_test_type	<i>O tipo do teste, com store, restore e preview no card trama/test.</i>
--------------	--

Description

O núcleo não registra tipos: quem registra é uma coleção (a data), com o id no namespace dela. Por isso o id é argumento.

Usage

```
tr_test_type(id)
```

Arguments

id id qualificado do tipo.

Value

um tr_type.

tr_text	<i>Texto "i18n-pronto": string única ou lista nomeada por idioma.</i>
---------	---

Description

Hoje quase tudo é escrito só em português, mas pressupostos e referências vão para o editor E para o site — e o site terá versão em inglês. Aceitar `list(pt = , en = )` desde já faz a tradução ser acréscimo de conteúdo, não mudança de API em todas as coleções. A escolha cai para pt (a língua em que o conteúdo nasce) e, na falta dela, para o primeiro idioma declarado: um texto só em inglês ainda é melhor que nenhum.

Usage

```
tr_text(x, lang = getOption("trama.lang", "pt"))
```

Arguments

x	string única, ou lista nomeada de strings únicas.
lang	código do idioma pedido.

Value

string única.

tr_theme	<i>Resolve um tema pelo nome, fora de um projeto.</i>
----------	---

Description

Para o console e para coleções: fn de nó recebe a definição já resolvida pelo plano, mas quem chama a função direto passa um nome ("claro"). Sem projeto, valem os temas embutidos.

Usage

```
tr_theme(tema = "padrão", settings = NULL)
```

Arguments

tema	Nome de um tema, "padrão", ou a definição já resolvida (lista).
settings	Settings do projeto; NULL usa os temas embutidos.

Details

Aceitar também a lista já resolvida é o que deixa a coleção ter UM caminho só: ela chama `tr_theme()` sem saber de onde veio o valor. A lista é revalidada porque no console qualquer um monta uma à mão, e fudno calado seria o mesmo silêncio que o manifesto recusa. Nome desconhecido não é erro aqui, pelo mesmo motivo do plano (documento não quebra por cosmético): volta o padrão marcado ausente, e quem chama decide se isso é erro.

Value

Lista com nome, os campos do tema e, se o nome não existe, `ausente = TRUE`.

tr_theme_names	<i>Nomes dos temas disponíveis.</i>
----------------	-------------------------------------

Description

Existe para a mensagem de erro de quem chama no console: "tema 'escuro' não existe" só ajuda se disser quais existem, e a coleção não pode copiar a lista dos embutidos — o projeto troca essa lista.

Usage

```
tr_theme_names(settings = NULL)
```

Arguments

settings	Settings do projeto; NULL usa os temas embutidos.
----------	---

Value

Vetor de nomes.

tr_transitions_read	<i>Lê as transições mineradas que uma coleção guarda no próprio pacote.</i>
---------------------	---

Description

O arquivo é o `transicoes.json` que `tools/sugestor/minerar.R` grava: um array `[{from, to, n}]`. Ausente devolve NULL em vez de erro, porque uma coleção sem exemplos não tem o que declarar, e isso não deve impedi-la de carregar.

Usage

```
tr_transitions_read(path)
```

Arguments

path	Caminho do JSON; costuma vir de <code>system.file()</code> , que devolve "" quando o arquivo não existe.
------	--

Value

`data.frame(from, to, n)` ou NULL.

tr_type	<i>Declara um tipo de dado (o que trafega numa porta).</i>
---------	--

Description

O tipo carrega a própria persistência (store/restore/ext). É aqui que "SpatRaster é ponteiro C++ e não sobrevive a saveRDS" se resolve — não no store, que permanece cego ao que guarda.

Usage

```
tr_type(
  id,
  version = 1L,
  label = NULL,
  color = "#64748b",
  store = NULL,
  restore = NULL,
  ext = "rds",
  preview = NULL,
  summary = NULL,
  report = NULL,
  tema = FALSE
)
```

Arguments

tema	O preview é um gráfico que segue o tema padrão do projeto.
------	--

Details

O insumo despachava serialização por classe R (`inherits(x, "SpatRaster")`), e isso não escala nem dentro de um domínio só: Heightmap e Field são ambos SpatRaster, e foi exatamente essa colisão que obrigou a existência de um nó identidade (`height_as_field`) só pra reetiquetar um no outro. Tipo é nominal; classe R é detalhe de implementação de um tipo.

`preview(x, ctx)` devolve um artefato — `tr_preview()` — cujo `render` é um id de string resolvido no front. Dispatch por id, nunca por `inherits()`: é o que permite uma coleção trazer visualização nova sem tocar no núcleo.

`report(x)` é opcional e diz como o valor aparece num relatório exportado (`tr_export_code()`): uma função EXPORTADA de pacote, que o documento chama sobre a saída (`tr_series_plot(x)` para uma série). Sem ele, o relatório imprime o objeto. Função e não `preview` porque o relatório roda sem o Trama.

Sem `store/restore`, cai em RDS — suficiente pra tipo que é objeto R comum (`data.frame`, lista, escalar).

`tema = TRUE` diz que o `preview` desenha um GRÁFICO com o tema "padrão". Aí o tema padrão do projeto entra na chave das saídas desse tipo (trocar o padrão redesenha o card) e vale como "padrão" enquanto o `preview` roda — sem isto o card de uma série ficava no tema embutido, escuro, num projeto todo claro.

tr_ui	<i>Monta a UI Shiny do editor: importmap, dependências e o <div> onde o React monta.</i>
-------	--

Description

Monta a UI Shiny do editor: importmap, dependências e o <div> onde o React monta.

Usage

```
tr_ui(project, plugins = list())
```

Arguments

plugins	Plugins do editor (ver tr_plugins()); tr_app() passa as instaladas.
---------	--

tr_use	<i>Carrega uma coleção num registro.</i>
--------	--

Description

Aceita o objeto tr_collection direto, ou o nome de um pacote que exporte trama_collection(). Colisão de id é erro alto e cedo — mas agora é improvável por construção, já que id é qualificado por coleção.

Usage

```
tr_use(
  collection,
  registry = .tr_default_registry,
  requires_of = .tr_collection_requires
)
```

Details

Pelo nome de pacote, as coleções de que ela depende (campo Config/trama/requires do DESCRIPTION, ver .tr_collection_requires()) entram antes, recursivamente, pulando as que o registro já tem. É isso que tira a ordem do trama.json do caminho: a validação de portas abaixo recusa tipo desconhecido, e sem isso um projeto que lista só trama.series quebrava por não ter pedido trama.data antes. requires_of existe para teste: injeta o grafo sem pacotes de verdade.

tr_value	<i>Valor de qualquer nó do grafo, no console.</i>
----------	---

Description

O caminho inverso do nível 1 (tr_fn(id)), e a outra metade do que impede a ferramenta de virar gaiola: dá pra inspecionar no REPL o que qualquer caixa da tela produziu, sem passar pela UI.

Usage

```
tr_value(
  doc,
  node,
  registry = .tr_default_registry,
  store,
  port = NULL,
  executor = tr_executor_sequential(),
  settings = NULL,
  ctx_extra = NULL
)
```

Details

Reaproveita o plano que tr_run() já devolve, em vez de montar outro. Um segundo plano sorteia um nonce novo pra nó volatile, e a chave consultada não seria a que acabou de ser gravada.

settings é o project\$settings: tema entra no hash, e sem ele o console resolveria o tema pelos embutidos e calcularia chave diferente da sessão — recomputando o que já está no cache, ou pior, mostrando outro gráfico.

ctx_extra está aqui porque pedir o valor de um nó no console RODA o grafo, e uma região de dez mil pontos é justamente o que alguém pede assim: sem o argumento, o único jeito de baixar a cadência de checkpoint num run de console seria não usar tr_value().

tr_when	<i>Mostra o param só sob condição sobre outros params do mesmo nó.</i>
---------	--

Description

tr_when(tr_param_num(0.05), metodo = c("holm", "bonferroni")) só exhibe o campo quando metodo vale um dos valores dados. Várias condições valem juntas (E). Os valores podem ser texto, lógico ou número.

Usage

```
tr_when(p, ...)
```

Arguments

p Um param de `tr_param()`.
 ... Condições nomeadas: <param> = <valores permitidos>.

Details

É puramente de interface: param escondido guarda o valor e continua indo para fn. Que cada nome citado seja outro param do nó é conferido em `tr_node()` (`tr_error_bad_when`), porque o param sozinho não conhece os irmãos.

Value

p com when preenchido.

tr_when_padrao	<i>Condição de <code>tr_when()</code> por padrão de texto: o param aparece quando o valor do outro casa com a expressão regular (sem diferenciar maiúsculas). Serve para o que uma lista de valores não expressa, como "o caminho é um link": <code>tr_when(p, path = tr_when_padrao("^https?://"))</code>.</i>
----------------	---

Description

Condição de `tr_when()` por padrão de texto: o param aparece quando o valor do outro casa com a expressão regular (sem diferenciar maiúsculas). Serve para o que uma lista de valores não expressa, como "o caminho é um link": `tr_when(p, path = tr_when_padrao("^https?://"))`.

Usage

```
tr_when_padrao(padrao)
```

Arguments

padrao Expressão regular, na sintaxe comum ao R e ao JavaScript.

Index

.tr_presentation_ops, 4

tr_adapter, 5

tr_adapter_for, 5

tr_add, 5

tr_app, 6

tr_bust, 7

tr_catalog, 7

tr_catalog_json, 7

tr_category, 8

tr_category(), 29

tr_cli, 9

tr_collection, 10

tr_compatible, 11

tr_dataset, 12

tr_dataset(), 11

tr_dataset_load, 13

tr_datasets, 13

tr_dependency_collection, 14

tr_dependency_core, 14

tr_dependency_importmap, 14

tr_doc, 15

tr_doc_apply, 15

tr_doc_json, 15

tr_doc_migrate, 16

tr_doc_migrate(), 30

tr_doc_parse, 16

tr_doc_read, 17

tr_doc_subset, 17

tr_doc_terminals, 17

tr_doc_validate, 18

tr_doc_write, 18

tr_errors, 19

tr_executor_pool, 19

tr_executor_sequential, 19

tr_export_code, 20

tr_export_code(), 63

tr_flow, 21

tr_flow(), 20

tr_flow_code, 22

tr_flow_code(), 20

tr_flow_doc, 22

tr_fn, 23

tr_galeria_captura, 23

tr_galeria_itens, 24

tr_galeria_itens(), 25

tr_galeria_sincronizar, 25

tr_get_node, 25

tr_get_type, 26

tr_handle_failed, 26

tr_help_test_card, 26

tr_icon, 27

tr_is_template, 27

tr_link, 27

tr_mcp, 28

tr_node, 28

tr_node(), 8, 30, 66

tr_node_raiox, 30

tr_op_semantic, 31

tr_param, 31

tr_param(), 66

tr_param_bool, 31

tr_param_col, 32

tr_param_enum, 33

tr_param_int, 33

tr_param_num, 34

tr_param_text, 34

tr_param_theme, 34

tr_plan, 35

tr_plan_blocked, 35

tr_plan_keys, 36

tr_plan_pending, 36

tr_plugins, 36

tr_plugins(), 64

tr_port, 37

tr_port_default, 37

tr_port_free, 38

tr_pressuposto, 38

tr_pressuposto(), 29

tr_preview, [38](#)
tr_progress, [39](#)
tr_project, [39](#)
tr_project_at, [39](#)
tr_project_flow, [40](#)
tr_project_gc, [40](#)
tr_project_new, [41](#)
tr_project_save, [41](#)
tr_project_set_galeria, [42](#)
tr_project_set_marca, [42](#)
tr_project_set_sugestoes, [43](#)
tr_project_set_themes, [43](#)
tr_provenance, [44](#)
tr_ref, [45](#)
tr_ref(), [29](#)
tr_registry, [45](#)
tr_registry_for, [46](#)
tr_replay, [46](#)
tr_retry, [46](#)
tr_run, [48](#)
tr_run_logger, [49](#)
tr_run_plan, [49](#)
tr_runs, [49](#)
tr_scheduler, [50](#)
tr_server, [50](#)
tr_set, [51](#)
tr_store, [51](#)
tr_store_gc, [52](#)
tr_store_get, [52](#)
tr_store_handle, [53](#)
tr_store_has, [53](#)
tr_store_path, [53](#)
tr_store_put, [54](#)
tr_store_put_error, [54](#)
tr_stream_command, [55](#)
tr_submit, [55](#)
tr_template, [56](#)
tr_template_dir, [56](#)
tr_template_json, [56](#)
tr_template_list, [57](#)
tr_template_op, [57](#)
tr_template_parse, [57](#)
tr_template_read, [58](#)
tr_template_save, [58](#)
tr_test, [58](#)
tr_test_table, [59](#)
tr_test_type, [60](#)
tr_text, [60](#)
tr_text(), [38](#)
tr_theme, [61](#)
tr_theme_names, [62](#)
tr_transitions_read, [62](#)
tr_transitions_read(), [10](#)
tr_type, [63](#)
tr_ui, [64](#)
tr_use, [64](#)
tr_value, [65](#)
tr_when, [65](#)
tr_when(), [66](#)
tr_when_padrao, [66](#)