Package 'AutoDeskR'

May 28, 2026

Type Package

Title An Interface to the 'AutoDesk' 'API' Platform

Description An interface to the 'AutoDesk' Platform Services ('APS') 'API' including the Authentication 'API' for obtaining 'OAuth2' tokens with expiry tracking, Data Management 'API' for managing buckets and objects across the platform's cloud services, Design Automation 'API' for running automated tasks on design files in the cloud, Model Derivative 'API' for translating design files into 'SVF', 'SVF2', 'OBJ', and 'STL' formats and extracting model data, Reality Capture 'API' for generating 3D models from photogrammetry image sets, and Viewer for rendering 2D and 3D models in 'Shiny' applications.

Version 0.5.0

URL https://aps.autodesk.com, https://github.com/paulgovan/AutoDeskR, https://paulgovan.github.io/AutoDeskR-Book/

BugReports https://github.com/paulgovan/AutoDeskR/issues

Depends R (>= 4.1.0)

License Apache License | file LICENSE

Imports curl, httr2, jsonlite, shiny

Encoding UTF-8

RoxygenNote 7.3.3

Suggests ellmer, knitr, mcptools, rmarkdown, testthat (>= 3.0.0), httptest2, tibble, withr

Config/testthat/edition 3

NeedsCompilation no

Author Paul Govan [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-1821-8492>)

Maintainer Paul Govan <paul.govan2@gmail.com>

Repository CRAN

Date/Publication 2026-05-28 14:20:15 UTC

Contents

`aps_error` *Create an APS API Error Condition.*

Description

Create a structured error condition for failures returned by the AutoDesk Platform Services (APS) API. Callers can use `tryCatch(..., aps_error = function(e) ...)` to handle these errors.

Usage

```
aps_error(message, status, body, call = sys.call(-1))
```

Arguments

`message`	A string. Human-readable error message.
`status`	An integer. HTTP status code.
`body`	A list. Parsed JSON body of the error response.
`call`	The call where the error occurred (internal use).

Value

A condition object of class `c("aps_error", "error", "condition")`.

Examples

```
## Not run:
tryCatch(
  makeBucket(token = "bad_token"),
  aps_error = function(e) cat("HTTP", e$status, "-", e$message)
)

## End(Not run)
```

`as_tibble.listBuckets` *Convert a listBuckets Response to a Tibble.*

Description

Requires the `tibble` package.

Usage

```
## S3 method for class 'listBuckets'
as_tibble(x, ...)
```

Arguments

`x`	A `listBuckets` response object.
`...`	Additional arguments (unused).

Value

A `tibble` with one row per bucket and columns `bucketKey`, `bucketOwner`, and `policyKey`.

Examples

```
## Not run:
library(tibble)
listBuckets(token = myToken) |> as_tibble()

## End(Not run)
```

`as_tibble.listObjects` *Convert a listObjects Response to a Tibble.*

Description

Requires the `tibble` package.

Usage

```
## S3 method for class 'listObjects'
as_tibble(x, ...)
```

Arguments

`x`	A `listObjects` response object.
`...`	Additional arguments (unused).

Value

A `tibble` with one row per object and columns `objectKey`, `objectId`, `size`, and `location`.

Examples

```
## Not run:
library(tibble)
listObjects(token = myToken, bucket = "mybucket") |> as_tibble()

## End(Not run)
```

autodeskr_mcp_tools *MCP tools for AutoDeskR*

Description

Returns a named list of `ellmer::tool()` objects that expose AutoDeskR functions to AI models via the Model Context Protocol (MCP). Credentials are read from the `APS_CLIENT_ID` and `APS_CLIENT_SECRET` environment variables; no token argument is required in any tool call.

Usage

```
autodeskr_mcp_tools()
```

Value

A named list of `ellmer::tool` objects covering Authentication, Data Management, Model Derivative, Design Automation, and Reality Capture.

Examples

```
## Not run:
Sys.setenv(APS_CLIENT_ID = "your_id", APS_CLIENT_SECRET = "your_secret")
tools <- autodeskr_mcp_tools()
mcptools::mcp_server(tools = tools)

## End(Not run)
```

checkBucket *Check the Status of an App-Managed Bucket.*

Description

Check the status of a recently created app-managed bucket using the Data Management API.

Usage

```
checkBucket(token = NULL, bucket = "mybucket")
```

Arguments

`token`	A string or `aps_token` object with `bucket:create`, `bucket:read`, and `data:write` scopes.
`bucket`	A string. Name of the bucket. Defaults to `mybucket`.

Value

An object containing the `bucketKey`, `bucketOwner`, and `createdDate`.

Examples

```
## Not run:
# Check the status of a bucket with the name "mybucket"
resp <- checkBucket(token = myToken, bucket = "mybucket")
resp

## End(Not run)
```

---

checkFile *Check the Status of a Translated File.*

Description

Check the status of a recently translated file using the Model Derivative API.

Usage

```
checkFile(urn = NULL, token = NULL)
```

Arguments

urn	A string. Source URN (objectId) for the file. Note the URN must be Base64 encoded. To encode the URN, see, for example, the `jsonlite::base64_enc` function.
token	A string or `aps_token` object with `data:read` and `data:write` scopes.

Examples

```
## Not run:
# Check the status of the translated "aerial.dwg" svf file
resp <- checkFile(urn = myEncodedUrn, token = myToken)
resp

## End(Not run)
```

---

checkPdf *Check the Status of a PDF WorkItem.*

Description

Check the status of a Design Automation WorkItem using the Design Automation API v3.

Usage

```
checkPdf(id = NULL, token = NULL, source = NULL, destination = NULL)
```

Arguments

id	A string. WorkItem ID returned by makePdf in resp$content$id.
token	A string or aps_token object with code:all scope.
source	Deprecated. Ignored with a warning.
destination	Deprecated. Ignored with a warning.

Value

An object containing the WorkItem id, status, and stats.

Examples

```
## Not run:
mySource <- "http://download.autodesk.com/us/samplefiles/acad/visualization_-_aerial.dwg"
myDestination <- "https://example.com/output/aerial.pdf"
resp <- makePdf(mySource, myDestination, token = myToken)
myWorkItemId <- resp$content$id

# Poll for completion
resp <- checkPdf(id = myWorkItemId, token = myToken)
resp

## End(Not run)
```

---

checkPhotoscene *Check Reality Capture Processing Progress.*

Description

Poll the processing status of a photoscene. Use waitForPhotoscene to block until processing completes.

Usage

```
checkPhotoscene(photoscene_id = NULL, token = NULL)
```

Arguments

photoscene_id	A string. Photoscene ID returned by createPhotoscene.
token	A string or aps_token object with data:read and data:write scopes.

Value

An object of class checkPhotoscene containing resp$content$photoscene$progress (percentage string) and resp$content$photoscene$progressmsg.

Examples

```
## Not run:
status <- checkPhotoscene(photoscene_id = myPhotosceneId, token = myToken)
status$content$photoscene$progress

## End(Not run)
```

`createPhotoscene` *Create a Photoscene for Reality Capture.*

Description

Create a new photoscene for photogrammetry processing using the Reality Capture API. Upload images with `uploadImages`, then start processing with `processPhotoscene`.

Usage

```
createPhotoscene(name = NULL, format = "rcm", token = NULL)
```

Arguments

`name`	A string. Name for the photoscene.
`format`	A string. Output format. One of `"rcm"`, `"rcs"`, `"obj"`, `"ortho"`, or `"report"`. Defaults to `"rcm"`.
`token`	A string or `aps_token` object with `data:read` and `data:write` scopes.

Value

An object of class `createPhotoscene` containing the `photosceneid` at `resp$content$photoscene$photosceneid`.

Examples

```
## Not run:
ps <- createPhotoscene(name = "my-scene", format = "obj", token = myToken)
myPhotosceneId <- ps$content$photoscene$photosceneid

## End(Not run)
```

deleteBucket *Delete an App-Managed Bucket.*

Description

Delete an app-managed bucket using the Data Management API.

Usage

```
deleteBucket(token = NULL, bucket = "mybucket")
```

Arguments

token	A string or aps_token object with bucket:delete scope.
bucket	A string. Name of the bucket to delete. Defaults to mybucket.

Value

An object containing the HTTP status code of the response.

Examples

```
## Not run:
# Delete a bucket named "mybucket"
resp <- deleteBucket(token = myToken, bucket = "mybucket")

## End(Not run)
```

deleteObject *Delete an Object from an App-Managed Bucket.*

Description

Delete an object from an app-managed bucket using the Data Management API.

Usage

```
deleteObject(token = NULL, bucket = "mybucket", object = NULL)
```

Arguments

token	A string or aps_token object with data:write scope.
bucket	A string. Name of the bucket. Defaults to mybucket.
object	A string. Key (name) of the object to delete.

Value

An object containing the HTTP status code of the response.

Examples

```
## Not run:
# Delete the "aerial.dwg" object from "mybucket"
resp <- deleteObject(token = myToken, bucket = "mybucket", object = "aerial.dwg")

## End(Not run)
```

`downloadFile` *Download a file locally.*

Description

Download a file from the AutoDesk Platform Services using the Model Derivative API.

Usage

```
downloadFile(urn = NULL, output_urn = NULL, token = NULL, destfile = NULL)
```

Arguments

`urn`	A string. Source URN (objectId) for the file. Note the URN must be Base64 encoded. To encode the URN, see, for example, the `jsonlite::base64_enc` function.
`output_urn`	A string. Output URN retrieved via `getOutputUrn`.
`token`	A string or `aps_token` object with `data:read` and `data:write` scopes.
`destfile`	A string. Local file path to save binary responses (e.g. downloaded geometry files). When `NULL` and the response is not JSON, a warning is issued and the raw bytes are returned.

Value

An object containing either parsed JSON content or, for binary responses, the path to the saved file.

Examples

```
## Not run:
# Download the "aerial.dwg" obj file
myEncodedOutputUrn <- jsonlite::base64_enc(myOutputUrn)
resp <- downloadFile(urn <- myEncodedUrn, output_urn <- myEncodedOutputUrn,
           token = myToken, destfile = "aerial.obj")

## End(Not run)
```

getData *Get the Geometry Data for a File.*

Description

Get the geometry of an uploaded file using the Model Derivative API.

Usage

```
getData(guid = NULL, urn = NULL, token = NULL)
```

Arguments

guid	A string. GUID retrieved via the getMetadata function.
urn	A string. Source URN (objectId) for the file. Note the URN must be Base64 encoded. To encode the URN, see, for example, the jsonlite::base64_enc function.
token	A string or aps_token object with data:read and data:write scopes.

Value

An object containing the geometry data for the selected file.

Examples

```
## Not run:
# Get the geometry data for the "aerial.dwg" svf file
resp <- getData(guid = myGuid, urn = myEncodedUrn, token = myToken)

## End(Not run)
```

getMetadata *Get the Metadata for a File.*

Description

Get the metadata of an uploaded file using the Model Derivative API.

Usage

```
getMetadata(urn = NULL, token = NULL)
```

Arguments

urn	A string. Source URN (objectId) for the file. Note the URN must be Base64 encoded. To encode the URN, see, for example, the `jsonlite::base64_enc` function.
token	A string or `aps_token` object with `data:read` and `data:write` scopes.

Value

An object containing the `type`, `name`, and `guid` of the file.

Examples

```
## Not run:
# Get the metadata for the "aerial.dwg" svf file
resp <- getMetadata(urn <- myEncodedUrn, token = myToken)
myGuid <- resp$content$data$metadata[[1]]$guid

## End(Not run)
```

`getObjectTree` *Get the Object Tree of a File.*

Description

Get the object tree of an uploaded file using the Model Derivative API.

Usage

```
getObjectTree(guid = NULL, urn = NULL, token = NULL)
```

Arguments

guid	A string. GUID retrieved via the `getMetadata` function.
urn	A string. Source URN (objectId) for the file. Note the URN must be Base64 encoded. To encode the URN, see, for example, the `jsonlite::base64_enc` function.
token	A string or `aps_token` object with `data:read` and `data:write` scopes.

Value

An object containing the object tree for the selected file.

Examples

```
## Not run:
# Get the object tree for the "aerial.dwg" svf file
resp <- getObjectTree(guid = myGuid, urn = myEncodedUrn, token = myToken)
resp

## End(Not run)
```

`getOutputUrn` *Get the Output URN for a File.*

Description

Get the output urn of a translated file using the Model Derivative API.

Usage

```
getOutputUrn(urn, token)
```

Arguments

`urn`	A string. Source URN (objectId) for the file. Note the URN must be Base64 encoded. To encode the URN, see, for example, the `jsonlite::base64_enc` function.
`token`	A string or `aps_token` object with `data:read` and `data:write` scopes.

Value

An object containing the `result`, `urn`, and additional activity information.

Examples

```
## Not run:
# Get the output urn for the "aerial.dwg" obj file
resp <- getOutputUrn(urn = myUrn, token = Sys.getenv("token"))
resp

## End(Not run)
```

getToken *Get a 2-Legged Token for Authentication.*

Description

Get a 2-legged token for OAuth-based authentication to the AutoDesk Platform Services (APS).

Usage

```
getToken(id = NULL, secret = NULL, scope = "data:write data:read")
```

Arguments

id	A string. Client ID for the app generated from the AutoDesk Dev Portal.
secret	A string. Client Secret for the app generated from the AutoDesk Dev Portal.
scope	A string. Space-separated list of required scopes. May be `user-profile:read`, `data:read`, `data:write`, `data:create`, `data:search`, `bucket:create`, `bucket:read`, `bucket:update`, `bucket:delete`, `code:all`, `account:read`, `account:write`, or a combination of these.

Value

An `aps_token` object containing the `access_token`, `token_type`, `expires_in`, and `expires_at`. The token can be passed directly to other AutoDeskR functions. Use `is_expired` to check whether the token needs refreshing. Legacy access via `resp$content$access_token` continues to work.

Examples

```
## Not run:
tok <- getToken(id = Sys.getenv("client_id"), secret = Sys.getenv("client_secret"),
         scope = "data:write data:read")
myToken <- tok$access_token
is_expired(tok)

## End(Not run)
```

is_expired *Check Whether an aps_token Has Expired.*

Description

Check Whether an aps_token Has Expired.

Usage

```
is_expired(token)
```

Arguments

token	An aps_token object returned by getToken.

Value

Logical. TRUE if the token has expired, FALSE otherwise.

Examples

```
## Not run:
tok <- getToken(id = Sys.getenv("client_id"), secret = Sys.getenv("client_secret"))
is_expired(tok)

## End(Not run)
```

---

listBuckets *List All App-Managed Buckets.*

Description

List all app-managed buckets using the Data Management API.

Usage

```
listBuckets(token = NULL, limit = 10, startAt = NULL, region = "US")
```

Arguments

token	A string or aps_token object with bucket:read scope.
limit	An integer. Maximum number of buckets to return. Defaults to 10.
startAt	A string. Bucket key to start the list from (for pagination). Defaults to NULL.
region	A string. Region filter. May be "US" or "EMEA". Defaults to "US".

Value

An object containing a list of bucket details.

Examples

```
## Not run:
# List all buckets
resp <- listBuckets(token = myToken)
resp$content$items

## End(Not run)
```

listObjects *List Objects in an App-Managed Bucket.*

Description

List objects stored in an app-managed bucket using the Data Management API.

Usage

```
listObjects(token = NULL, bucket = "mybucket", limit = 10)
```

Arguments

token	A string or aps_token object with data:read scope.
bucket	A string. Name of the bucket. Defaults to mybucket.
limit	An integer. Maximum number of objects to return. Defaults to 10.

Value

An object containing a list of objects in the bucket.

Examples

```
## Not run:
# List objects in "mybucket"
resp <- listObjects(token = myToken, bucket = "mybucket")
resp$content$items

## End(Not run)
```

makeBucket *Make a Bucket for an App.*

Description

Make an app-based bucket for storage of design files using the Data Management API.

Usage

```
makeBucket(token = NULL, bucket = "mybucket", policy = "transient")
```

Arguments

token	A string or aps_token object with bucket:create, bucket:read, and data:write scopes.
bucket	A string. Unique bucket name. Defaults to mybucket.
policy	A string. May be transient, temporary, or persistent.

Value

An object containing the `bucketKey`, `bucketOwner`, and `createdDate`.

Examples

```
## Not run:
# Make a transient bucket with the name "mybucket"
resp <- makeBucket(token = myToken, bucket = "mybucket", policy = "transient")

## End(Not run)
```

---

makePdf *Convert a DWG to a PDF.*

Description

Convert a publicly accessible DWG file to a publicly accessible PDF using the Design Automation API v3.

Usage

```
makePdf(source = NULL, destination = NULL, token = NULL)
```

Arguments

`source`	A string. Publicly accessible web address of the input DWG file.
`destination`	A string. Publicly accessible web address for the output PDF file.
`token`	A string or `aps_token` object with `code:all` scope.

Value

An object containing the WorkItem `id`, `status`, and `stats`. Use `id` with `checkPdf` to poll for completion, or use `waitForWorkItem` to block until done.

Examples

```
## Not run:
mySource <- "http://download.autodesk.com/us/samplefiles/acad/visualization_-_aerial.dwg"
myDestination <- "https://example.com/output/aerial.pdf"
resp <- makePdf(mySource, myDestination, token = myToken)
myWorkItemId <- resp$content$id

## End(Not run)
```

processPhotoscene *Start Reality Capture Processing.*

Description

Initiate photogrammetry processing for a photoscene that has had images uploaded via `uploadImages`.

Usage

```
processPhotoscene(photoscene_id = NULL, token = NULL)
```

Arguments

photoscene_id	A string. Photoscene ID returned by `createPhotoscene`.
token	A string or `aps_token` object with `data:read` and `data:write` scopes.

Value

An object of class `processPhotoscene` containing the processing response.

Examples

```
## Not run:
proc <- processPhotoscene(photoscene_id = myPhotosceneId, token = myToken)

## End(Not run)
```

translateObj *Translate a File into OBJ Format.*

Description

Translate an uploaded file into OBJ format using the Model Derivative API.

Usage

```
translateObj(urn = NULL, token = NULL)
```

Arguments

urn	A string. Source URN (objectId) for the file. Note the URN must be Base64 encoded. To encode the URN, see, for example, the `jsonlite::base64_enc` function.
token	A string or `aps_token` object with `data:read` and `data:write` scopes.

Value

An object containing the `result`, `urn`, and additional activity information.

Examples

```
## Not run:
# Translate the "aerial.dwg" file into an obj file
resp <- translateObj(urn = myEncodedUrn, token = myToken)

## End(Not run)
```

`translateStl` *Translate a File into STL Format.*

Description

Translate an uploaded file into STL format using the Model Derivative API.

Usage

```
translateStl(urn = NULL, token = NULL)
```

Arguments

`urn`	A string. Source URN (objectId) for the file. Note the URN must be Base64 encoded. To encode the URN, see, for example, the `jsonlite::base64_enc` function.
`token`	A string or `aps_token` object with `data:read` and `data:write` scopes.

Value

An object containing the `result`, `urn`, and additional activity information.

Examples

```
## Not run:
# Translate the "aerial.dwg" file into an stl file
resp <- translateStl(urn = myEncodedUrn, token = myToken)

## End(Not run)
```

translateSvf — *Translate a File into SVF Format.*

Description

Translate an uploaded file into SVF format using the Model Derivative API.

Usage

```
translateSvf(urn = NULL, token = NULL)
```

Arguments

urn	A string. Source URN (objectId) for the file. Note the URN must be Base64 encoded. To encode the URN, see, for example, the `jsonlite::base64_enc` function.
token	A string or `aps_token` object with `data:read` and `data:write` scopes.

Value

An object containing the `result`, `urn`, and additional activity information.

Examples

```
## Not run:
# Translate the "aerial.dwg" file into a svf file
myEncodedUrn <- jsonlite::base64_enc(myUrn)
resp <- translateSvf(urn = myEncodedUrn, token = myToken)

## End(Not run)
```

translateSvf2 — *Translate a File into SVF2 Format.*

Description

Translate an uploaded file into SVF2 format using the Model Derivative API. SVF2 is the next-generation viewer format: approximately 30 SVF and faster to load in the Autodesk Viewer. Use it in place of `translateSvf` for new projects.

Usage

```
translateSvf2(urn = NULL, token = NULL, views = c("2d", "3d"))
```

Arguments

urn	A string. Source URN (objectId) for the file. Note the URN must be Base64 encoded. To encode the URN, see, for example, the jsonlite::base64_enc function.
token	A string or aps_token object with data:read and data:write scopes.
views	A character vector. Views to generate. Defaults to c("2d", "3d").

Value

An object containing the result, urn, and additional activity information.

Examples

```
## Not run:
# Translate the "aerial.dwg" file into SVF2 format
myEncodedUrn <- jsonlite::base64_enc(myUrn)
resp <- translateSvf2(urn = myEncodedUrn, token = myToken)

## End(Not run)
```

---

uploadFile *Upload a File to an App-Managed Bucket.*

Description

Upload a design file to an app-managed bucket using the Data Management API.

Usage

```
uploadFile(file = NULL, token = NULL, bucket = "mybucket")
```

Arguments

file	A string. File path.
token	A string or aps_token object with bucket:create, bucket:read, and data:write scopes.
bucket	A string. Unique bucket name. Defaults to mybucket.

Value

An object containing the bucketKey, objectId (i.e. urn), objectKey (i.e. file name), size, contentType (i.e. "application/octet-stream"), location. and other content information.

Examples

```
## Not run:
# Upload the "aerial.dwg" file to "mybucket"
resp <- uploadFile(file = system.file("inst/samples/aerial.dwg", package = "AutoDeskR"),
           token = myToken, bucket = "mybucket")
myUrn <- resp$content$objectId

## End(Not run)
```

`uploadFileSigned` *Upload a File Using Signed S3 URLs.*

Description

Upload a design file of any size to an app-managed bucket using the signed S3 URL approach recommended by AutoDesk Platform Services (APS). Unlike `uploadFile`, this function supports files larger than 100 MB.

Usage

```
uploadFileSigned(file = NULL, token = NULL, bucket = "mybucket")
```

Arguments

`file`	A string. File path.
`token`	A string or `aps_token` object with `data:write` scope.
`bucket`	A string. Unique bucket name. Defaults to `mybucket`.

Value

An object containing the finalized upload response with `bucketKey`, `objectId`, `objectKey`, `size`, and `location`.

Examples

```
## Not run:
# Upload a large file using signed S3 URLs
resp <- uploadFileSigned(
  file   = "path/to/large_model.rvt",
  token  = myToken,
  bucket = "mybucket"
)
myUrn <- resp$content$objectId

## End(Not run)
```

uploadImages *Upload Images to a Photoscene.*

Description

Upload one or more image files to an existing photoscene for Reality Capture processing.

Usage

```
uploadImages(photoscene_id = NULL, files = NULL, token = NULL)
```

Arguments

photoscene_id	A string. Photoscene ID returned by createPhotoscene.
files	A character vector. Local file paths to image files (JPEG or PNG).
token	A string or aps_token object with data:read and data:write scopes.

Value

An object of class uploadImages containing the upload response.

Examples

```
## Not run:
imgs <- uploadImages(
  photoscene_id = myPhotosceneId,
  files         = c("img1.jpg", "img2.jpg", "img3.jpg"),
  token         = myToken
)

## End(Not run)
```

viewer3D *Launch the Viewer.*

Description

Launch the Viewer.

Usage

```
viewer3D(urn = NULL, token = NULL, viewerType = "header")
```

Arguments

urn	A string. Source URN (objectId) for the file. Note the URN must be Base64 encoded. To encode the URN, see, for example, the `jsonlite::base64_enc` function.
token	A string. Token generated with `getToken` function with `data:read` scope.
viewerType	A string. The type of viewer to instantiate. Either "header" for the default viewer, "headless" for a viewer without toolbar or panels, or "vr" to enter WebVR mode on a mobile device.

Examples

```
## Not run:
# View the "aerial.dwg" file in the AutoDesk viewer
myEncodedUrn <- jsonlite::base64_enc(myUrn)
viewer3D(urn <- myEncodedUrn, token = myToken)

## End(Not run)
```

---

viewerUI *UI Module Function.*

Description

UI Module Function.

Usage

```
viewerUI(id, urn = NULL, token = NULL, viewerType = "header")
```

Arguments

id	A string. A namespace for the module.
urn	A string. Source URN (objectId) for the file. Note the URN must be Base64 encoded. To encode the URN, see, for example, the `jsonlite::base64_enc` function.
token	A string. Token generated with `getToken` function with `data:read` scope.
viewerType	A string. The type of viewer to instantiate. Either "header" for the default viewer or "headless" for a viewer without toolbar or panels.

Examples

```
## Not run:
ui <- function(request) {
 shiny::fluidPage(
   viewerUI("pg", myEncodedUrn, myToken)
 )
}
```

```
server <- function(input, output, session) {
}
shiny::shinyApp(ui, server)

## End(Not run)
```

`waitForFile` *Wait for a Model Derivative Translation to Complete.*

Description

Polls `checkFile` at a fixed interval until the translation reaches a terminal state (`"success"`, `"failed"`, or `"timeout"`).

Usage

```
waitForFile(urn, token, interval = 5, timeout = 300, verbose = TRUE)
```

Arguments

`urn`	A string. Base64-encoded source URN.
`token`	A string or `aps_token` object with `data:read` scope.
`interval`	Seconds between polls. Defaults to `5`.
`timeout`	Maximum seconds to wait before aborting. Defaults to `300`.
`verbose`	If `TRUE` (default), prints a message after each poll.

Value

The final `checkFile` response object.

Examples

```
## Not run:
myEncodedUrn <- jsonlite::base64_enc(myUrn)
resp <- translateSvf(urn = myEncodedUrn, token = myToken)
done <- waitForFile(urn = myEncodedUrn, token = myToken)
done$content$status

## End(Not run)
```

`waitForPhotoscene` *Wait for Reality Capture Processing to Complete.*

Description

Polls `checkPhotoscene` at a fixed interval until processing reaches 100% or an error occurs.

Usage

```
waitForPhotoscene(
  photoscene_id,
  token,
  interval = 30,
  timeout = 1800,
  verbose = TRUE
)
```

Arguments

`photoscene_id`	A string. Photoscene ID returned by `createPhotoscene`.
`token`	A string or `aps_token` object.
`interval`	Seconds between polls. Defaults to `30`.
`timeout`	Maximum seconds to wait before aborting. Defaults to `1800` (30 minutes).
`verbose`	If `TRUE` (default), prints a message after each poll.

Value

The final `checkPhotoscene` response object.

Examples

```
## Not run:
ps   <- createPhotoscene("my-scene", token = myToken)
id   <- ps$content$photoscene$photosceneid
imgs <- uploadImages(id, c("img1.jpg", "img2.jpg"), myToken)
proc <- processPhotoscene(id, myToken)
done <- waitForPhotoscene(id, myToken)
done$content$photoscene$progress

## End(Not run)
```

`waitForWorkItem` *Wait for a Design Automation WorkItem to Complete.*

Description

Polls `checkPdf` at a fixed interval until the WorkItem reaches a terminal state (any status other than `"inprogress"` or `"pending"`).

Usage

```
waitForWorkItem(id, token, interval = 5, timeout = 300, verbose = TRUE)
```

Arguments

`id`	A string. WorkItem ID from `makePdf()$content$id`.
`token`	A string or `aps_token` object with `code:all` scope.
`interval`	Seconds between polls. Defaults to `5`.
`timeout`	Maximum seconds to wait before aborting. Defaults to `300`.
`verbose`	If `TRUE` (default), prints a message after each poll.

Value

The final `checkPdf` response object.

Examples

```
## Not run:
resp <- makePdf(source = mySource, destination = myDest, token = myToken)
done <- waitForWorkItem(id = resp$content$id, token = myToken)
done$content$status

## End(Not run)
```

Index