Config Item#

Base class for an instantiable object. Instantiate the target component and return the instance. object Return a boolean flag to indicate whether the object should be instantiated. bool Scan all the available classes and functions in the MONAI package and map them with the module paths in a table. It's used to locate the module path for provided component name. excludes (Union[str, Sequence[str], None]) – if any string of the excludes exists in the full module name, don't import this module. Get the full module name of the class or function with specified name. If target component name exists in multiple packages or modules, return a list of full module names. name (str) – name of the expected class or function. Union[List[str], str, None] Subclass of monai.bundle.ConfigItem, this class uses a dictionary with string keys to represent a component of class or function and supports instantiation. Currently, three special keys (strings surrounded by _) are defined and interpreted beyond the regular literals: class or function identifier of the python module, specified by "_target_", indicating a build-in python class or function such as "LoadImageDict", or a full module name, such as "monai.transforms.LoadImageDict". "_requires_" (optional): specifies reference IDs (string starts with "@") or ConfigExpression of the dependencies for this ConfigComponent object. These dependencies will be evaluated/instantiated before this object is instantiated. It is useful when the component doesn't explicitly depend on the other ConfigItems via its arguments, but requires the dependencies to be instantiated/evaluated beforehand. "_disabled_" (optional): a flag to indicate whether to skip the instantiation. "_desc_" (optional): free text descriptions of the component for code readability. Other fields in the config content are input arguments to the python module. config (Any) – content of a config item. id (str) – name of the current config item, defaults to empty string. locator (Optional[ComponentLocator]) – a ComponentLocator to convert a module name string into the actual python module. if None, a ComponentLocator(excludes=excludes) will be used. excludes (Union[str, Sequence[str], None]) – if locator is None, create a new ComponentLocator with excludes. See also: monai.bundle.ComponentLocator. Instantiate component based on self.config content. The target component must be a class or a function, otherwise, return None. kwargs – args to override / add the config args when instantiation. object Utility function used in instantiate() to check whether to skip the instantiation. bool Check whether this config represents a class or function that is to be instantiated. config (Any) – input config content to check. bool Utility function used in instantiate() to resolve the arguments from current config content. Resolve the target module name from current config content. The config content must have "_target_" key. Subclass of monai.bundle.ConfigItem, the ConfigItem represents an executable expression (execute based on eval(), or import the module to the globals if it's an import statement). See also https://docs.python.org/3/library/functions.html#eval. For example: config (Any) – content of a config item. id (str) – name of current config item, defaults to empty string. globals (Optional[Dict]) – additional global context to evaluate the string. Execute the current config content and return the result if it is expression, based on Python eval(). For more details: https://docs.python.org/3/library/functions.html#eval. globals (Optional[Dict]) – besides self.globals, other global symbols used in the expression at runtime. locals (Optional[Dict]) – besides globals, may also have some local symbols used in the expression at runtime. Check whether the config is an executable expression string. Currently, a string starts with "$" character is interpreted as an expression. config (Union[Dict, List, str]) – input config content to check. bool Check whether the config is an import statement (a special case of expression). config (Union[Dict, List, str]) – input config content to check. bool Basic data structure to represent a configuration item. A ConfigItem instance can optionally have a string id, so that other items can

refer to it. It has a build-in config property to store the configuration object. config (Any) – content of a config item, can be objects of any types, a configuration resolver may interpret the content to generate a configuration object. id (str) – name of the current config item, defaults to empty string. Get the config content of current config item. Get the ID name of current config item, useful to identify config items during parsing. str Replace the content of self.config with new config. A typical usage is to modify the initial config content at runtime. config (Any) – content of a ConfigItem.

Reference Resolver#

Utility class to manage a set of ConfigItem and resolve the references between them. This class maintains a set of ConfigItem objects and their associated IDs. The IDs must be unique within this set. A string in ConfigItem starting with @ will be treated as a reference to other ConfigItem objects by ID. Since ConfigItem may have a nested dictionary or list structure, the reference string may also contain a # character to refer to a substructure by key indexing for a dictionary or integer indexing for a list. In this class, resolving references is essentially substitution of the reference strings with the corresponding python objects. A typical workflow of resolving references is as follows: Add multiple ConfigItem objects to the ReferenceResolver by add_item(). If it is instantiable, instantiate it and cache the class instance in resolved_content. If it is an expression, evaluate it and save the value in resolved_content. Substitute the reference strings with the corresponding objects. items (Optional[Sequence[ConfigItem]]) – ConfigItem``s to resolve, this could be added later with ``add_item(). Add a ConfigItem to the resolver. item (ConfigItem) – a ConfigItem. Recursively search all the content of input config item to get the ids of references. References mean: the IDs of other config items ("@XXX" in this config item), or the sub-item in the config is instantiable, or the sub-item in the config is expression. For dict and list, recursively check the sub-items. config – input config content to search. id (str) – ID name for the input config item. refs (Optional[Dict[str, int]]) – dict of the ID name and count of found references, default to None. Dict[str, int] Get the ConfigItem by id. If resolve=True, the returned item will be resolved, that is, all the reference strings are substituted by the corresponding ConfigItem objects. id (str) – id of the expected config item. resolve (bool) – whether to resolve the item if it is not resolved, default to False. kwargs – keyword arguments to pass to _resolve_one_item(). Currently support instantiate and eval_expr. Both are defaulting to True. Get the resolved ConfigItem by id. id (str) – id name of the expected item. kwargs – keyword arguments to pass to _resolve_one_item(). Currently support instantiate, eval_expr and default. instantiate and eval_expr are defaulting to True, default is the target config item if the id is not in the config content, must be a ConfigItem object. Match regular expression for the input string to find the references. The reference string starts with "@", like: "@XXX#YYY#ZZZ". value (str) – input value to match regular expression. Dict[str, int] Clear all the added ConfigItem and all the resolved content. With all the references in refs, update the input config content with references and return the new config. config – input config content to update. id (str) – ID name for the input config. refs (Optional[Dict]) – all the referring content with ids, default to None. Match regular expression for the input string to update content with the references. The reference part starts with "@", like: "@XXX#YYY#ZZZ". References dictionary must contain the referring IDs as keys. value (str) – input value to match regular expression. refs (Dict) – all the referring components with ids as keys, default to None. str

Config Parser#

The primary configuration parser. It traverses a structured config (in the form of nested Python dict or list), creates ConfigItem, and assign unique IDs according to the structures. This class provides convenient access to the set of ConfigItem of the config by ID. A typical workflow of config parsing is as follows: Initialize ConfigParser with the config source. Call get_parsed_content() to get expected component with id. config (Optional[Any]) – input config source to parse. excludes (Union[str, Sequence[str], None]) – when importing modules to instantiate components, excluding components from modules specified in excludes. globals (Union[Dict[str, Any], None, bool]) – pre-import packages as global variables to ConfigExpression, so that expressions, for example, "$monai.data.list_data_collate" can use monai modules. The current supported globals and alias names are {"monai": "monai", "torch": "torch", "np": "numpy", "numpy": "numpy"}. These are MONAI's minimal dependencies. Additional packages could be included with globals={"itk": "itk"}. Set it to False to disable self.globals module importing. See also monai.bundle.ConfigItem monai.bundle.scripts.run Export the config content to the specified file path (currently support JSON and YAML files). config (Dict) – source config content to export. filepath (Union[str, PathLike]) – target file path to save. fmt – format of config content, currently support "json" and "yaml". kwargs – other arguments for json.dump or yaml.safe_dump, depends on the file format. Get the config by id. id (str) – id to specify the expected position. See also __getitem__(). default (Optional[Any]) – default value to return if the specified id is invalid. Get the parsed result of ConfigItem with the specified id. If the item is ConfigComponent and instantiate=True, the result is the instance. If the item is ConfigExpression and eval_expr=True, the result is the evaluated output. Else, the result is the configuration content of ConfigItem. id (str) – id of the ConfigItem, "#" in id are interpreted as special characters to go one level further into the nested structures. Use digits indexing from "0" for list or other strings for dict. For example: "xform#5", "net#channels". "" indicates the entire self.config. kwargs – additional keyword arguments to be passed to _resolve_one_item. Currently support lazy (whether to retain the current config cache, default to True), instantiate (whether to instantiate the ConfigComponent, default to True) and eval_expr (whether to evaluate the ConfigExpression, default to True), default (the default config item if the id is not in the config content). Load config file with specified file path (currently support JSON and YAML files). filepath (Union[str, PathLike]) – path of target file to load, supported postfixes: .json, .yml, .yaml. kwargs – other arguments for json.load or `yaml.safe_load, depends on the file format. Load config files into a single config dict. The latter config file in the list will override or add the former config file. "#" in the config keys are interpreted as special characters to go one level further into the nested structures. files (Union[str, PathLike, Sequence[Union[str, PathLike]], dict]) – path of target files to load, supported postfixes: .json, .yml, .yaml. kwargs – other arguments for json.load or `yaml.safe_load, depends on the file format. Dict Recursively resolve self.config to replace the macro tokens with target content. Then recursively parse the config source, add every item as ConfigItem to the reference resolver. reset (bool) – whether to reset the reference_resolver before parsing. Defaults to True. Read the config from specified JSON or YAML file. The config content in the self.config dictionary. f (Union[str, PathLike, Sequence[Union[str, PathLike]], Dict]) – filepath of the config file, the content must be a dictionary, if providing a list of files, wil merge the content of them. if providing a dictionary directly, use it as config. kwargs – other arguments for json.load or yaml.safe_load, depends on the file format. Read the metadata from specified JSON or YAML file. The metadata as a dictionary will be stored at self.config["_meta_"]. f (Union[str, PathLike, Sequence[Union[str, PathLike]], Dict]) – filepath of the metadata file, the content must

be a dictionary, if providing a list of files, will merge the content of them. if providing a dictionary directly, use it as metadata. kwargs – other arguments for json.load or yaml.safe_load, depends on the file format. Recursively resolve self.config to replace the relative ids with absolute ids, for example, @##A means A in the upper level. and replace the macro tokens with target content, The macro tokens are marked as starting with “%”, can be from another structured file, like: "%default_net", "%/data/config.json#net". To simplify the reference or macro tokens ID in the nested config content, it’s available to use relative ID name which starts with the ID_SEP_KEY, for example, “@#A” means A in the same level, @##A means A in the upper level. It resolves the relative ids to absolute ids. For example, if the input data is: It will resolve B to {“key”: “@A”, “value1”: 2, “value2”: “%B#value1”, “value3”: [3, 4, “@B#value3#1”]}. id (str) – id name for current config item to compute relative id. value (str) – input value to resolve relative ids. str Set config by id. config (Any) – config to set at location id. id (str) – id to specify the expected position. See also __setitem__(). recursive (bool) – if the nested id doesn’t exist, whether to recursively create the nested items in the config. default to True. for the nested id, only support dict for the missing section. Split src string into two parts: a config file path and component id. The file path should end with (json|yaml|yml). The component id should be separated by # if it exists. If no path or no id, return “”. src (str) – source string to split. Tuple[str, str] Set the id and the corresponding config content in pairs, see also __setitem__(). For example, parser.update({"train#epoch": 100, "train#lr": 0.02}) pairs (Dict[str, Any]) – dictionary of id and config pairs.

Scripts#

Export the model checkpoint to the given filepath with metadata and config included as JSON files. Typical usage examples: net_id (Optional[str]) – ID name of the network component in the config, it must be torch.nn.Module. filepath (Union[str, PathLike, None]) – filepath to export, if filename has no extension it becomes .ts. ckpt_file (Optional[str]) – filepath of the model checkpoint to load. meta_file (Union[str, Sequence[str], None]) – filepath of the metadata file, if it is a list of file paths, the content of them will be merged. config_file (Union[str, Sequence[str], None]) – filepath of the config file to save in TorchScript model and extract network information, the saved key in the TorchScript model is the config filename without extension, and the saved config value is always serialized in JSON format no matter the original file format is JSON or YAML. it can be a single file or a list of files. if None, must be provided in args_file. key_in_ckpt (Optional[str]) – for nested checkpoint like {“model”: XXX, “optimizer”: XXX, …}, specify the key of model weights. if not nested checkpoint, no need to set. args_file (Optional[str]) – a JSON or YAML file to provide default values for meta_file, config_file, net_id and override pairs. so that the command line inputs can be simplified. override – id-value pairs to override or add the corresponding config content. e.g. --_meta#network_data_format#inputs#image#num_channels 3. download bundle from the specified source or url. The bundle should be a zip file and it will be extracted after downloading. This function refers to: https://pytorch.org/docs/stable/_modules/torch/hub.html Typical usage examples: name (Optional[str]) – bundle name. If None and url is None, it must be provided in args_file. for example: “spleen_ct_segmentation”, “prostate_mri_anatomy” in model-zoo: Project-MONAI/model-zoo. “monai_brats_mri_segmentation” in ngc: https://catalog.ngc.nvidia.com/models?filters=&orderBy;=scoreDESC&query;=monai. version (Optional[str]) – version name of the target bundle to download, like: “0.1.0”. If None, will download the latest version. bundle_dir (Union[str, PathLike, None]) – target directory to store the downloaded data. Default is bundle subfolder under

torch.hub.get_dir(). source (str) – storage location name. This argument is used when url is None. In default, the value is achieved from the environment variable BUNDLE_DOWNLOAD_SRC, and it should be “ngc” or “github”. repo (Optional[str]) – repo name. This argument is used when url is None and source is “github”. If used, it should be in the form of “repo_owner/repo_name/release_tag”. url (Optional[str]) – url to download the data. If not None, data will be downloaded directly and source will not be checked. If name is None, filename is determined by monai.apps.utils._basename(url). remove_prefix (Optional[str]) – This argument is used when source is “ngc”. Currently, all ngc bundles have the monai_ prefix, which is not existing in their model zoo contrasts. In order to maintain the consistency between these two sources, remove prefix is necessary. Therefore, if specified, downloaded folder name will remove the prefix. progress (bool) – whether to display a progress bar. args_file (Optional[str]) – a JSON or YAML file to provide default values for all the args in this function. so that the command line inputs can be simplified. Load model weights or TorchScript module of a bundle. name (str) – bundle name. If None and url is None, it must be provided in args_file. for example: “spleen_ct_segmentation”, “prostate_mri_anatomy” in model-zoo: Project-MONAI/model-zoo. “monai_brats_mri_segmentation” in ngc: https://catalog.ngc.nvidia.com/models?filters=&orderBy;=scoreDESC&query;=monai. version (Optional[str]) – version name of the target bundle to download, like: “0.1.0”. If None, will download the latest version. model_file (Optional[str]) – the relative path of the model weights or TorchScript module within bundle. If None, “models/model.pt” or “models/model.ts” will be used. load_ts_module (bool) – a flag to specify if loading the TorchScript module. bundle_dir (Union[str, PathLike, None]) – directory the weights/TorchScript module will be loaded from. Default is bundle subfolder under torch.hub.get_dir(). source (str) – storage location name. This argument is used when model_file is not existing locally and need to be downloaded first. In default, the value is achieved from the environment variable BUNDLE_DOWNLOAD_SRC, and it should be “ngc” or “github”. repo (Optional[str]) – repo name. This argument is used when url is None and source is “github”. If used, it should be in the form of “repo_owner/repo_name/release_tag”. remove_prefix (Optional[str]) – This argument is used when source is “ngc”. Currently, all ngc bundles have the monai_ prefix, which is not existing in their model zoo contrasts. In order to maintain the consistency between these two sources, remove prefix is necessary. Therefore, if specified, downloaded folder name will remove the prefix. progress (bool) – whether to display a progress bar when downloading. device (Optional[str]) – target device of returned weights or module, if None, prefer to “cuda” if existing. key_in_ckpt (Optional[str]) – for nested checkpoint like {“model”: XXX, “optimizer”: XXX, …}, specify the key of model weights. if not nested checkpoint, no need to set. config_files (Sequence[str]) – extra filenames would be loaded. The argument only works when loading a TorchScript module, see _extra_files in torch.jit.load for more details. net_name (Optional[str]) – if not None, a corresponding network will be instantiated and load the achieved weights. This argument only works when loading weights. net_kwargs – other arguments that are used to instantiate the network class defined by net_name. If load_ts_module is False and net_name is None, return model weights. If load_ts_module is False and net_name is not None,return an instantiated network that loaded the weights. If load_ts_module is True, return a triple that include a TorchScript module,the corresponding metadata dict, and extra files dict. please check monai.data.load_net_with_metadata for more details. If load_ts_module is False and net_name is None, return model weights. return an instantiated network that loaded the weights. the corresponding metadata dict, and extra files dict. please check monai.data.load_net_with_metadata for more details. Get all bundles names (and the latest versions) that are stored in the release of specified repository with the provided

tag. The default values of arguments correspond to the release of MONAI model zoo. In order to increase the rate limits of calling Github APIs, you can input your personal access token. Please check the following link for more details about rate limiting: https://docs.github.com/en/rest/overview/resources-in-the-rest-api#rate-limiting The following link shows how to create your personal access token: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token repo (str) – it should be in the form of “repo_owner/repo_name/”. tag (str) – the tag name of the release. auth_token (Optional[str]) – github personal access token. a list of tuple in the form of (bundle name, latest version). Get all information (include “id”, “name”, “size”, “download_count”, “browser_download_url”, “created_at”, “updated_at”) of a bundle with the specified bundle name and version. In order to increase the rate limits of calling Github APIs, you can input your personal access token. Please check the following link for more details about rate limiting: https://docs.github.com/en/rest/overview/resources-in-the-rest-api#rate-limiting The following link shows how to create your personal access token: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token bundle_name (str) – bundle name. version (Optional[str]) – version name of the target bundle, if None, the latest version will be used. repo (str) – it should be in the form of “repo_owner/repo_name/”. tag (str) – the tag name of the release. auth_token (Optional[str]) – github personal access token. a dictionary that contains the bundle’s information. Get the latest version, as well as all existing versions of a bundle that is stored in the release of specified repository with the provided tag. In order to increase the rate limits of calling Github APIs, you can input your personal access token. Please check the following link for more details about rate limiting: https://docs.github.com/en/rest/overview/resources-in-the-rest-api#rate-limiting The following link shows how to create your personal access token: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token bundle_name (str) – bundle name. repo (str) – it should be in the form of “repo_owner/repo_name/”. tag (str) – the tag name of the release. auth_token (Optional[str]) – github personal access token. a dictionary that contains the latest version and all versions of a bundle. Specify meta_file and config_file to run monai bundle components and workflows. Typical usage examples: runner_id (Union[str, Sequence[str], None]) – ID name of the expected config expression to run, can also be a list of IDs to run in order. meta_file (Union[str, Sequence[str], None]) – filepath of the metadata file, if it is a list of file paths, the content of them will be merged. config_file (Union[str, Sequence[str], None]) – filepath of the config file, if None, must be provided in args_file. if it is a list of file paths, the content of them will be merged. logging_file (Optional[str]) – config file for logging module in the program, default to None. for more details: https://docs.python.org/3/library/logging.config.html#logging.config.fileConfig. tracking (Union[str, dict, None]) – enable the experiment tracking feature at runtime with optionally configurable and extensible. if “mlflow”, will add MLFlowHandler to the parsed bundle with default logging settings, if other string, treat it as file path to load the logging settings, if dict, treat it as logging settings, otherwise, use all the default settings. will patch the target config content with tracking handlers and the top-level items of configs. example of customized settings: tracking = { "handlers_id": { "trainer": {"id": "train#trainer", "handlers": "train#handlers"}, "validator": {"id": "evaluate#evaluator", "handlers": "evaluate#handlers"}, "evaluator": {"id": "evaluator", "handlers": "handlers"}, }, "configs": { "tracking_uri": "", "experiment_name": "monai_experiment", "run_name": None, "is_not_rank0": ("$torch.distributed.is_available() and torch.distributed.is_initialized() and torch.distributed.get_rank() > 0"), "trainer": { "_target_": "MLFlowHandler", "_disabled_": "@is_not_rank0", "tracking_uri": "@tracking_uri", "experiment_name":

"@experiment_name", "run_name": "@run_name", "iteration_log": True, "output_transform": "$monai.handlers.from_engine(['loss'], first=True)", "close_on_complete": True, }, "validator": { "_target_": "MLFlowHandler", "_disabled_": "@is_not_rank0", "tracking_uri": "@tracking_uri", "experiment_name": "@experiment_name", "run_name": "@run_name", "iteration_log": False, }, "evaluator": { "_target_": "MLFlowHandler", "_disabled_": "@is_not_rank0", "tracking_uri": "@tracking_uri", "experiment_name": "@experiment_name", "run_name": "@run_name", "iteration_log": False, "close_on_complete": True, }, }, }, enable the experiment tracking feature at runtime with optionally configurable and extensible. if “mlflow”, will add MLFlowHandler to the parsed bundle with default logging settings, if other string, treat it as file path to load the logging settings, if dict, treat it as logging settings, otherwise, use all the default settings. will patch the target config content with tracking handlers and the top-level items of configs. example of customized settings: args_file (Optional[str]) – a JSON or YAML file to provide default values for runner_id, meta_file, config_file, logging, and override pairs. so that the command line inputs can be simplified. override – id-value pairs to override or add the corresponding config content. e.g. --net#input_chns 42. Verify the provided metadata file based on the predefined schema. metadata content must contain the schema field for the URL of schema file to download. The schema standard follows: http://json-schema.org/. meta_file (Union[str, Sequence[str], None]) – filepath of the metadata file to verify, if None, must be provided in args_file. if it is a list of file paths, the content of them will be merged. filepath (Union[str, PathLike, None]) – file path to store the downloaded schema. create_dir (Optional[bool]) – whether to create directories if not existing, default to True. hash_val (Optional[str]) – if not None, define the hash value to verify the downloaded schema file. hash_type (Optional[str]) – if not None, define the hash type to verify the downloaded schema file. Defaults to “md5”. args_file (Optional[str]) – a JSON or YAML file to provide default values for all the args in this function. so that the command line inputs can be simplified. kwargs – other arguments for jsonschema.validate(). for more details: https://python-jsonschema.readthedocs.io/en/stable/validate/#jsonschema.validate. Verify the input and output data shape and data type of network defined in the metadata. Will test with fake Tensor data according to the required data shape in metadata. Typical usage examples: net_id (Optional[str]) – ID name of the network component to verify, it must be torch.nn.Module. meta_file (Union[str, Sequence[str], None]) – filepath of the metadata file to get network args, if None, must be provided in args_file. if it is a list of file paths, the content of them will be merged. config_file (Union[str, Sequence[str], None]) – filepath of the config file to get network definition, if None, must be provided in args_file. if it is a list of file paths, the content of them will be merged. device (Optional[str]) – target device to run the network forward computation, if None, prefer to “cuda” if existing. p (Optional[int]) – power factor to generate fake data shape if dim of expected shape is “x**p”, default to 1. n (Optional[int]) – multiply factor to generate fake data shape if dim of expected shape is “x*n”, default to 1. any (Optional[int]) – specified size to generate fake data shape if dim of expected shape is “*”, default to 1. args_file (Optional[str]) – a JSON or YAML file to provide default values for net_id, meta_file, config_file, device, p, n, any, and override pairs. so that the command line inputs can be simplified. override – id-value pairs to override or add the corresponding config content. e.g. --_meta#network_data_format#inputs#image#num_channels 3. Initialise a new bundle directory with some default configuration files and optionally network weights. Typical usage example: bundle_dir (Union[str, PathLike]) – directory name to create, must not exist but parent direct must exist ckpt_file (Union[str, PathLike, None]) – optional checkpoint file to copy into bundle network (Optional[Module]) – if given instead of ckpt_file this network’s weights will be stored in bundle dataset_license

(bool) – if True, a default license file called “data_license.txt” will be produced. This file is required if there are any license conditions stated for data your bundle uses. metadata_str (Union[Dict, str, None]) – optional metadata string to write to bundle, if not given a default will be used. inference_str (Union[Dict, str, None]) – optional inference string to write to bundle, if not given a default will be used. previous Federated Learning next Transforms © Copyright MONAI Consortium.