Text Compare
Produced: 01/09/2025 03:25:35 PM

Mode: Differences
Left file: /tmp/before.py Right file: /tmp/after.py

		<>	558	class CharmedDashboard:	
			559	"""A helper class for handling dashboards on the requirer (Grafana) side."""	
			560		
			561	@classmethod	
558	def _convert_dashboard_fields(content: str, inject_dropdowns: bool = True) -> str:		562	def _convert_dashboard_fields(cls, content: str, inject_dropdowns: bool = True) -> str:	
559	"""Make sure values are present for Juju topology.		563	"""Make sure values are present for Juju topology.	
561	Inserts Juju topology variables and selectors into the template, as well as	<>	565	Inserts Juju topology variables and selectors into the template, as well as	
562	a variable for Prometheus.		566	a variable for Prometheus.	
563	"""		567	"""	
564	dict_content = json.loads(content)		568	dict_content = json.loads(content)	
565	datasources = {}		569	datasources = {}	
566	existing_templates = False		570	existing_templates = False	
568	template_dropdowns = (	<>	572	template_dropdowns = (	
569	TOPOLOGY_TEMPLATE_DROPDOWNS + DATASOURCE_TEMPLATE_DROPDOWNS # type: ignore		573	TOPOLOGY_TEMPLATE_DROPDOWNS + DATASOURCE_TEMPLATE_DROPDOWNS # type: ignore	
570	if inject_dropdowns		574	if inject_dropdowns	
571	else DATASOURCE_TEMPLATE_DROPDOWNS		575	else DATASOURCE_TEMPLATE_DROPDOWNS	
572	)		576	)	
574	# If the dashboard has __inputs, get the names to replace them. These are stripped	<>	578	# If the dashboard has __inputs, get the names to replace them. These are stripped	
575	# from reactive dashboards in GrafanaDashboardAggregator, but charm authors in		579	# from reactive dashboards in GrafanaDashboardAggregator, but charm authors in	
576	# newer charms may import them directly from the marketplace		580	# newer charms may import them directly from the marketplace	
577	if "__inputs" in dict_content:		581	if "__inputs" in dict_content:	
578	for field in dict_content["__inputs"]:		582	for field in dict_content["__inputs"]:	
579	if "type" in field and field["type"] == "datasource":		583	if "type" in field and field["type"] == "datasource":	
580	datasources[field["name"]] = field["pluginName"].lower()		584	datasources[field["name"]] = field["pluginName"].lower()	
581	del dict_content["__inputs"]		585	del dict_content["__inputs"]	
583	# If no existing template variables exist, just insert our own	<>	587	# If no existing template variables exist, just insert our own	
584	if "templating" not in dict_content:		588	if "templating" not in dict_content:	
585	dict_content["templating"] = {"list": list(template_dropdowns)} # type: ignore		589	dict_content["templating"] = {"list": list(template_dropdowns)} # type: ignore	
586	else:		590	else:	
587	# Otherwise, set a flag so we can go back later		591	# Otherwise, set a flag so we can go back later	
588	existing_templates = True		592	existing_templates = True	

```
589         for template_value in dict_content["templating"]["list"]:
590             # Build a list of `datasource_name`: `datasource_type` mappings
591             # The "query" field is actually "prometheus", "loki", "influxdb", etc
592             if "type" in template_value and template_value["type"] == "datasource":
593                 datasources[template_value["name"]] = template_value["query"].lower()
```

```
593             for template_value in dict_content["templating"]["list"]:
594                 # Build a list of `datasource_name`: `datasource_type` mappings
595                 # The "query" field is actually "prometheus", "loki", "influxdb", etc
596                 if "type" in template_value and template_value["type"] == "datasource":
597                     datasources[template_value["name"]] = template_value["query"].lower()
```

```
595         # Put our own variables in the template
596         for d in template_dropdowns:  # type: ignore
597             if d not in dict_content["templating"]["list"]:
598                 dict_content["templating"]["list"].insert(0, d)
```

<>

```
599             # Put our own variables in the template
600             for d in template_dropdowns:  # type: ignore
601                 if d not in dict_content["templating"]["list"]:
602                     dict_content["templating"]["list"].insert(0, d)
```

```
600     dict_content = _replace_template_fields(dict_content, datasources, existing_templates)
601     return json.dumps(dict_content)
```

<>

```
604         dict_content = cls._replace_template_fields(dict_content, datasources, existing_templates)
605         return json.dumps(dict_content)
```

```
603
604 def _replace_template_fields(  # noqa: C901
605     dict_content: dict, datasources: dict, existing_templates: bool
606 ) -> dict:
607     """Make templated fields get cleaned up afterwards.
```

<>

```
607     @classmethod
608     def _replace_template_fields(  # noqa: C901
609         cls, dict_content: dict, datasources: dict, existing_templates: bool
610     ) -> dict:
611         """Make templated fields get cleaned up afterwards.
```

```
609     If existing datasource variables are present, try to substitute them.
610     """
611     replacements = {"loki": "${lokids}", "prometheus": "${prometheusds}"}
612     used_replacements = []  # type: List[str]
```

<>

```
613         If existing datasource variables are present, try to substitute them.
614         """
615         replacements = {"loki": "${lokids}", "prometheus": "${prometheusds}"}
616         used_replacements = []  # type: List[str]
```

```
614     # If any existing datasources match types we know, or we didn't find
615     # any templating variables at all, template them.
616     if datasources or not existing_templates:
617         panels = dict_content.get("panels", {})
618         if panels:
619             dict_content["panels"] = _template_panels(
620                 panels, replacements, used_replacements, existing_templates, datasources
621             )
```

<>

```
618         # If any existing datasources match types we know, or we didn't find
619         # any templating variables at all, template them.
620         if datasources or not existing_templates:
621             panels = dict_content.get("panels", {})
622             if panels:
623                 dict_content["panels"] = cls._template_panels(
624                     panels, replacements, used_replacements, existing_templates, datasources
625                 )
```

```
623         # Find panels nested under rows
624         rows = dict_content.get("rows", {})
625         if rows:
626             for row_idx, row in enumerate(rows):
627                 if "panels" in row.keys():
628                     rows[row_idx]["panels"] = _template_panels(
```

<>

```
627             # Find panels nested under rows
628             rows = dict_content.get("rows", {})
629             if rows:
630                 for row_idx, row in enumerate(rows):
631                     if "panels" in row.keys():
632                         rows[row_idx]["panels"] = cls._template_panels(
```

```
629                         row["panels"],
630                         replacements,
631                         used_replacements,
632                         existing_templates,
633                         datasources,
634                     )
```

```
633                         row["panels"],
634                         replacements,
635                         used_replacements,
636                         existing_templates,
637                         datasources,
638                     )
```

```
636             dict_content["rows"] = rows
```

<>

```
640             dict_content["rows"] = rows
```

```
638     # Finally, go back and pop off the templates we stubbed out
639     deletions = []
640     for tmpl in dict_content["templating"]["list"]:
641         if tmpl["name"] and tmpl["name"] in used_replacements:
642             deletions.append(tmpl)
```

<>

```
642         # Finally, go back and pop off the templates we stubbed out
643         deletions = []
644         for tmpl in dict_content["templating"]["list"]:
645             if tmpl["name"] and tmpl["name"] in used_replacements:
646                 deletions.append(tmpl)
```

```
644     for d in deletions:
645         dict_content["templating"]["list"].remove(d)
```

<>

```
648         for d in deletions:
649             dict_content["templating"]["list"].remove(d)
```

```
647     return dict_content
```

<>

```
651         return dict_content
```

```
649
650 def _template_panels(
```

<>

```
653     @classmethod
654     def _template_panels(
655         cls,
656         panels: dict,
657         replacements: dict,
658         used_replacements: list,
659         existing_templates: bool,
660         datasources: dict,
661     ) -> dict:
662         """Iterate through a `panels` object and template it appropriately."""
663         # Go through all the panels. If they have a datasource set, AND it's one
664         # that we can convert to ${lokids} or ${prometheusds}, by stripping off the
665         # ${} templating and comparing the name to the list we built, replace it,
666         # otherwise, leave it alone.
667         #
668         for panel in panels:
669             if "datasource" not in panel or not panel.get("datasource"):
670                 continue
671             if not existing_templates:
672                 datasource = panel.get("datasource")
673                 if isinstance(datasource, str):
674                     if "loki" in datasource:
675                         panel["datasource"] = "${lokids}"
676                     elif "grafana" in datasource:
677                         continue
```

```
651     panels: dict,
652     replacements: dict,
653     used_replacements: list,
654     existing_templates: bool,
655     datasources: dict,
656 ) -> dict:
657     """Iterate through a `panels` object and template it appropriately."""
658     # Go through all the panels. If they have a datasource set, AND it's one
659     # that we can convert to ${lokids} or ${prometheusds}, by stripping off the
660     # ${} templating and comparing the name to the list we built, replace it,
661     # otherwise, leave it alone.
662     #
663     for panel in panels:
664         if "datasource" not in panel or not panel.get("datasource"):
665             continue
666         if not existing_templates:
667             datasource = panel.get("datasource")
668             if isinstance(datasource, str):
669                 if "loki" in datasource:
670                     panel["datasource"] = "${lokids}"
671                 elif "grafana" in datasource:
672                     continue
```

```
673                 else:                                                   678                     else:
674                     panel["datasource"] = "${prometheusds}"             679                         panel["datasource"] = "${prometheusds}"
675             elif isinstance(datasource, dict):                          680                 elif isinstance(datasource, dict):
676                 # In dashboards exported by Grafana 9, datasource type is dict   681                     # In dashboards exported by Grafana 9, datasource type is dict
677                 dstype = datasource.get("type", "")                     682                     dstype = datasource.get("type", "")
678                 if dstype == "loki":                                    683                     if dstype == "loki":
679                     panel["datasource"]["uid"] = "${lokids}"            684                         panel["datasource"]["uid"] = "${lokids}"
680                 elif dstype == "prometheus":                            685                     elif dstype == "prometheus":
681                     panel["datasource"]["uid"] = "${prometheusds}"      686                         panel["datasource"]["uid"] = "${prometheusds}"
682                 else:                                                   687                     else:
683                     logger.debug("Unrecognized datasource type '%s'; skipping", dstype)   688                         logger.debug("Unrecognized datasource type '%s'; skipping", dstype)
684                     continue                                            689                         continue
685             else:                                                       690                 else:
686                 logger.error("Unknown datasource format: skipping")     691                     logger.error("Unknown datasource format: skipping")
687                 continue                                                692                     continue
688         else:                                                           693             else:
689             if isinstance(panel["datasource"], str):                    694                 if isinstance(panel["datasource"], str):
690                 if panel["datasource"].lower() in replacements.values():   695                     if panel["datasource"].lower() in replacements.values():
691                     # Already a known template variable                 696                         # Already a known template variable
692                     continue                                            697                         continue
693                 # Strip out variable characters and maybe braces        698                     # Strip out variable characters and maybe braces
694                 ds = re.sub(r"(\$|\{|\})", "", panel["datasource"])     699                     ds = re.sub(r"(\$|\{|\})", "", panel["datasource"])

696                 if ds not in datasources.keys():                    <>  701                     if ds not in datasources.keys():
697                     # Unknown, non-templated datasource, potentially a Grafana builtin   702                         # Unknown, non-templated datasource, potentially a Grafana builtin
698                     continue                                            703                         continue

700                 replacement = replacements.get(datasources[ds], "")  <>  705                     replacement = replacements.get(datasources[ds], "")
701                 if replacement:                                         706                     if replacement:
702                     used_replacements.append(ds)                        707                         used_replacements.append(ds)
703                 panel["datasource"] = replacement or panel["datasource"]   708                     panel["datasource"] = replacement or panel["datasource"]
704             elif isinstance(panel["datasource"], dict):                 709                 elif isinstance(panel["datasource"], dict):
705                 dstype = panel["datasource"].get("type", "")            710                     dstype = panel["datasource"].get("type", "")
706                 if panel["datasource"].get("uid", "").lower() in replacements.values():   711                     if panel["datasource"].get("uid", "").lower() in replacements.values():
707                     # Already a known template variable                 712                         # Already a known template variable
708                     continue                                            713                         continue
709                 # Strip out variable characters and maybe braces        714                     # Strip out variable characters and maybe braces
710                 ds = re.sub(r"(\$|\{|\})", "", panel["datasource"].get("uid", ""))   715                     ds = re.sub(r"(\$|\{|\})", "", panel["datasource"].get("uid", ""))
```

```
712                 if ds not in datasources.keys():
713                     # Unknown, non-templated datasource,
    potentially a Grafana builtin
714                     continue
716                 replacement = replacements.get(datasources[ds],
    "")
717                 if replacement:
718                     used_replacements.append(ds)
719                     panel["datasource"]["uid"] = replacement
720             else:
721                 logger.error("Unknown datasource format:
    skipping")
722                 continue
723     return panels
725
726 def _inject_labels(content: str, topology: dict, transformer:
    "CosTool") -> str:
727     """Inject Juju topology into panel expressions via CosTool.
729     A dashboard will have a structure approximating:
730         {
731             "__inputs": [],
732             "templating": {
733                 "list": [
734                     {
735                         "name": "prometheusds",
736                         "type": "prometheus"
737                     }
738                 ]
739             },
740             "panels": [
741                 {
742                     "foo": "bar",
743                     "targets": [
744                         {
745                             "some": "field",
746                             "expr": "up{job="foo"}"
747                         },
748                         {
749                             "some_other": "field",
750                             "expr":
    "sum(http_requests_total{instance="$foo"}[5m])}
751                         }
752                     ],
753                     "datasource": "${someds}"
754                 }
```

```
717                     if ds not in datasources.keys():
718                         # Unknown, non-templated datasource, potentially
    a Grafana builtin
719                         continue
721                     replacement = replacements.get(datasources[ds], "")
722                     if replacement:
723                         used_replacements.append(ds)
724                         panel["datasource"]["uid"] = replacement
725                 else:
726                     logger.error("Unknown datasource format: skipping")
727                     continue
728         return panels
730     @classmethod
731     def _inject_labels(cls, content: str, topology: dict, transformer:
    "CosTool") -> str:
732         """Inject Juju topology into panel expressions via CosTool.
734         A dashboard will have a structure approximating:
735             {
736                 "__inputs": [],
737                 "templating": {
738                     "list": [
739                         {
740                             "name": "prometheusds",
741                             "type": "prometheus"
742                         }
743                     ]
744                 },
745                 "panels": [
746                     {
747                         "foo": "bar",
748                         "targets": [
749                             {
750                                 "some": "field",
751                                 "expr": "up{job="foo"}"
752                             },
753                             {
754                                 "some_other": "field",
755                                 "expr":
    "sum(http_requests_total{instance="$foo"}[5m])}
756                             }
757                         ],
758                         "datasource": "${someds}"
759                     }
```

```
755             ]
756         }
```

```
758     `templating` is used elsewhere in this library, but the structure is not rigid. It is
759     not guaranteed that a panel will actually have any targets (it could be a "spacer" with
760     no datasource, hence no expression). It could have only one target. It could have multiple
761     targets. It could have multiple targets of which only one has an `expr` to evaluate. We need
762     to try to handle all of these concisely.
```

```
764     `cos-tool` (`github.com/canonical/cos-tool` as a Go module in general)
765     does not know "Grafana-isms", such as using `[$_variable]` to modify the query from the user
766     interface, so we add placeholders (as `5y`, since it must parse, but a dashboard looking for
767     five years for a panel query would be unusual).
```

```
769     Args:
770         content: dashboard content as a string
771         topology: a dict containing topology values
772         transformer: a 'CosTool' instance
773     Returns:
774         dashboard content with replaced values.
775     """
776     dict_content = json.loads(content)
```

```
778     if "panels" not in dict_content.keys():
779         return json.dumps(dict_content)
```

```
781     # Go through all the panels and inject topology labels
782     # Panels may have more than one 'target' where the expressions live, so that must be
783     # accounted for. Additionally, `promql-transform` does not necessarily gracefully handle
784     # expressions with range queries including variables. Exclude these.
785     #
786     # It is not a certainty that the `datasource` field will necessarily reflect the type, so
787     # operate on all fields.
788     panels = dict_content["panels"]
789     topology_with_prefix = {"juju_{}".format(k): v for k, v in topology.items()}
```

```
791     # We need to use an index so we can insert the changed element back later
```

```
760             ]
761         }
```

```
763     `templating` is used elsewhere in this library, but the structure is not rigid. It is
764     not guaranteed that a panel will actually have any targets (it could be a "spacer" with
765     no datasource, hence no expression). It could have only one target. It could have multiple
766     targets. It could have multiple targets of which only one has an `expr` to evaluate. We need
767     to try to handle all of these concisely.
```

```
769     `cos-tool` (`github.com/canonical/cos-tool` as a Go module in general)
770     does not know "Grafana-isms", such as using `[$_variable]` to modify the query from the user
771     interface, so we add placeholders (as `5y`, since it must parse, but a dashboard looking for
772     five years for a panel query would be unusual).
```

```
774     Args:
775         content: dashboard content as a string
776         topology: a dict containing topology values
777         transformer: a 'CosTool' instance
778     Returns:
779         dashboard content with replaced values.
780     """
781     dict_content = json.loads(content)
```

```
783     if "panels" not in dict_content.keys():
784         return json.dumps(dict_content)
```

```
786     # Go through all the panels and inject topology labels
787     # Panels may have more than one 'target' where the expressions live, so that must be
788     # accounted for. Additionally, `promql-transform` does not necessarily gracefully handle
789     # expressions with range queries including variables. Exclude these.
790     #
791     # It is not a certainty that the `datasource` field will necessarily reflect the type, so
792     # operate on all fields.
793     panels = dict_content["panels"]
794     topology_with_prefix = {"juju_{}".format(k): v for k, v in topology.items()}
```

```
796     # We need to use an index so we can insert the changed element back later
```

```
792     for panel_idx, panel in enumerate(panels):
793         if not isinstance(panel, dict):
794             continue
```

```
796         # Use the index to insert it back in the same location
797         panels[panel_idx] = _modify_panel(panel,
topology_with_prefix, transformer)
```

```
799     return json.dumps(dict_content)
```

```
801
802 def _modify_panel(panel: dict, topology: dict, transformer:
"CosTool") -> dict:
803     """Inject Juju topology into panel expressions via CosTool.
```

```
805     Args:
806         panel: a dashboard panel as a dict
807         topology: a dict containing topology values
808         transformer: a 'CosTool' instance
809     Returns:
810         the panel with injected values
811     """
812     if "targets" not in panel.keys():
813         return panel
```

```
815     # Pre-compile a regular expression to grab values from inside
of []
816     range_re = re.compile(r"\[(?P<value>.*?)\]")
817     # Do the same for any offsets
818     offset_re = re.compile(r"offset\s+(?P<value>-?\s*[$\w]+)")
```

```
820     known_datasources = {"${prometheusds}": "promql", "${lokids}":
"logql"}
```

```
822     targets = panel["targets"]
```

```
824     # We need to use an index so we can insert the changed element
back later
825     for idx, target in enumerate(targets):
826         # If there's no expression, we don't need to do anything
827         if "expr" not in target.keys():
828             continue
829         expr = target["expr"]
```

```
831         if "datasource" not in panel.keys():
832             continue
```

```
834         if isinstance(panel["datasource"], str):
835             if panel["datasource"] not in known_datasources:
836                 continue
837             querytype = known_datasources[panel["datasource"]]
838         elif isinstance(panel["datasource"], dict):
```

```
797         for panel_idx, panel in enumerate(panels):
798             if not isinstance(panel, dict):
799                 continue
```

```
801             # Use the index to insert it back in the same location
802             panels[panel_idx] = cls._modify_panel(panel,
topology_with_prefix, transformer)
```

```
804         return json.dumps(dict_content)
```

```
806     @classmethod
807     def _modify_panel(cls, panel: dict, topology: dict, transformer:
"CosTool") -> dict:
808         """Inject Juju topology into panel expressions via CosTool.
```

```
810         Args:
811             panel: a dashboard panel as a dict
812             topology: a dict containing topology values
813             transformer: a 'CosTool' instance
814         Returns:
815             the panel with injected values
816         """
817         if "targets" not in panel.keys():
818             return panel
```

```
820         # Pre-compile a regular expression to grab values from inside of
[]
821         range_re = re.compile(r"\[(?P<value>.*?)\]")
822         # Do the same for any offsets
823         offset_re = re.compile(r"offset\s+(?P<value>-?\s*[$\w]+)")
```

```
825         known_datasources = {"${prometheusds}": "promql", "${lokids}":
"logql"}
```

```
827         targets = panel["targets"]
```

```
829         # We need to use an index so we can insert the changed element
back later
830         for idx, target in enumerate(targets):
831             # If there's no expression, we don't need to do anything
832             if "expr" not in target.keys():
833                 continue
834             expr = target["expr"]
```

```
836             if "datasource" not in panel.keys():
837                 continue
```

```
839             if isinstance(panel["datasource"], str):
840                 if panel["datasource"] not in known_datasources:
841                     continue
842                 querytype = known_datasources[panel["datasource"]]
843             elif isinstance(panel["datasource"], dict):
```

```
839             if panel["datasource"]["uid"] not in known_datasources:
840                 continue
841             querytype = known_datasources[panel["datasource"]["uid"]]
842         else:
843             logger.error("Unknown datasource format: skipping")
844             continue
```

```
844                 if panel["datasource"]["uid"] not in known_datasources:
845                     continue
846                 querytype = known_datasources[panel["datasource"]["uid"]]
847             else:
848                 logger.error("Unknown datasource format: skipping")
849                 continue
```

```
846         # Capture all values inside `[]` into a list which we'll iterate over later to
847         # put them back in-order. Then apply the regex again and replace everything with
848         # `[5y]` so promql/parser will take it.
849         #
850         # Then do it again for offsets
851         range_values = [m.group("value") for m in range_re.finditer(expr)]
852         expr = range_re.sub(r"[5y]", expr)
```

<>

```
851             # Capture all values inside `[]` into a list which we'll iterate over later to
852             # put them back in-order. Then apply the regex again and replace everything with
853             # `[5y]` so promql/parser will take it.
854             #
855             # Then do it again for offsets
856             range_values = [m.group("value") for m in range_re.finditer(expr)]
857             expr = range_re.sub(r"[5y]", expr)
```

```
854         offset_values = [m.group("value") for m in offset_re.finditer(expr)]
855         expr = offset_re.sub(r"offset 5y", expr)
856         # Retrieve the new expression (which may be unchanged if there were no label
857         # matchers in the expression, or if tt was unable to be parsed like logql. It's
858         # virtually impossible to tell from any datasource "name" in a panel what the
859         # actual type is without re-implementing a complete dashboard parser, but no
860         # harm will some from passing invalid promql -- we'll just get the original back.
861         #
862         replacement = transformer.inject_label_matchers(expr, topology, querytype)
```

<>

```
859             offset_values = [m.group("value") for m in offset_re.finditer(expr)]
860             expr = offset_re.sub(r"offset 5y", expr)
861             # Retrieve the new expression (which may be unchanged if there were no label
862             # matchers in the expression, or if tt was unable to be parsed like logql. It's
863             # virtually impossible to tell from any datasource "name" in a panel what the
864             # actual type is without re-implementing a complete dashboard parser, but no
865             # harm will some from passing invalid promql -- we'll just get the original back.
866             #
867             replacement = transformer.inject_label_matchers(expr, topology, querytype)
```

```
864         if replacement == target["expr"]:
865             # promql-tranform caught an error. Move on
866             continue
```

<>

```
869             if replacement == target["expr"]:
870                 # promql-transform caught an error. Move on
871                 continue
```

```
868         # Go back and substitute values in [] which were pulled out
869         # Enumerate with an index... again. The same regex is ok, since it will still match
870         # `[(.*?)]`, which includes `[5y]`, our placeholder
871         for i, match in enumerate(range_re.finditer(replacement)):
872             # Replace one-by-one, starting from the left. We build the string back with
```

<>

```
873             # Go back and substitute values in [] which were pulled out
874             # Enumerate with an index... again. The same regex is ok, since it will still match
875             # `[(.*?)]`, which includes `[5y]`, our placeholder
876             for i, match in enumerate(range_re.finditer(replacement)):
877                 # Replace one-by-one, starting from the left. We build the string back with
```

873	# `str.replace(string_to_replace, replacement_value, count)`. Limit the count		878	# `str.replace(string_to_replace, replacement_value, count)`. Limit the count
874	# to one, since we are going through one-by-one through the list we saved earlier		879	# to one, since we are going through one-by-one through the list we saved earlier
875	# in `range_values`.		880	# in `range_values`.
876	replacement = replacement.replace(		881	replacement = replacement.replace(
877	"[{}]".format(match.group("value")),		882	"[{}]".format(match.group("value")),
878	"[{}]".format(range_values[i]),		883	"[{}]".format(range_values[i]),
879	1,		884	1,
880	)		885	)
882	for i, match in enumerate(offset_re.finditer(replacement)):	<>	887	for i, match in enumerate(offset_re.finditer(replacement)):
883	# Replace one-by-one, starting from the left. We build the string back with		888	# Replace one-by-one, starting from the left. We build the string back with
884	# `str.replace(string_to_replace, replacement_value, count)`. Limit the count		889	# `str.replace(string_to_replace, replacement_value, count)`. Limit the count
885	# to one, since we are going through one-by-one through the list we saved earlier		890	# to one, since we are going through one-by-one through the list we saved earlier
886	# in `range_values`.		891	# in `range_values`.
887	replacement = replacement.replace(		892	replacement = replacement.replace(
888	"offset {}".format(match.group("value")),		893	"offset {}".format(match.group("value")),
889	"offset {}".format(offset_values[i]),		894	"offset {}".format(offset_values[i]),
890	1,		895	1,
891	)		896	)
893	# Use the index to insert it back in the same location	<>	898	# Use the index to insert it back in the same location
894	targets[idx]["expr"] = replacement		899	targets[idx]["expr"] = replacement
896	panel["targets"] = targets	<>	901	panel["targets"] = targets
897	return panel		902	return panel
1442	content = _convert_dashboard_fields(content, inject_dropdowns)	<>	1447	content = CharmedDashboard._convert_dashboard_fields(content, inject_dropdowns)
1445	content = _inject_labels(content, topology, self._tranformer)	<>	1450	content = CharmedDashboard._inject_labels(content, topology, self._tranformer)