TEAMCENTER

Product Configurator — Deployment and Administration

Teamcenter 2412

SIEMENS

About Siemens Digital Industries Software

Siemens Digital Industries Software is a global leader in the growing field of product lifecycle management (PLM), manufacturing operations management (MOM), and electronic design automation (EDA) software, hardware, and services. Siemens works with more than 100,000 customers, leading the digitalization of their planning and manufacturing processes. At Siemens Digital Industries Software, we blur the boundaries between industry domains by integrating the virtual and physical, hardware and software, design and manufacturing worlds. With the rapid pace of innovation, digitalization is no longer tomorrow’s idea. We take what the future promises tomorrow and make it real for our customers today. Where today meets tomorrow. Our culture encourages creativity, welcomes fresh thinking and focuses on growth, so our people, our business, and our customers can achieve their full potential.

Support Center: **support.sw.siemens.com**

Send Feedback on Documentation: **support.sw.siemens.com/doc_feedback_form**

Contents

Setting up Product Configurator 1-1

Installing Product Configurator (mandatory)

Install Product Configurator using TEM 2-1
Install Product Configurator using Deployment Center 2-1
Delete configuration snapshot data after upgrading or applying patches 2-2
Update the fnd0impactingobjects property on SVRs to improve performance after an upgrade 2-3

Deploying Product Configurator services (optional)

Product Configurator service components 3-1
What is the Product Configurator service? 3-1
Deployment scenarios for Product Configurator service components 3-3

Installing the Product Configurator service (optional)

Task flow for installing the Product Configurator service 4-1
Prerequisites for installing the Product Configurator service 4-1
Install the Product Configurator service using TEM 4-2
Install the Product Configurator service using Deployment Center 4-3
Specify the URL and the port to connect Teamcenter to the machine on which microservices are running 4-5
Specify configurator as the service name to send tasks from Teamcenter to Product Configurator 4-5
Verify and troubleshoot the Product Configurator service installation 4-6

Configuring the Product Configurator service

Modify mandatory and optional preferences 5-1
Configure the Product Configurator service 5-3
View advanced information about the Product Configurator service including logs and requests 5-7
All about Docker for the Product Configurator service 5-10

Administering Product Configurator

About administering Product Configurator 6-1
Specify the validation mode for the Variant Configuration view 6-2
Allow or disallow users from creating and editing variant option data 6-2
Administering validation mode in Active Workspace using preferences 6-2
Set business object constants 6-8
Using name or ID as the primary business attribute 6-8
Use Name or ID as the primary business attribute 6-8

Set name or ID as the primary business attribute 6-11
Utilities and preferences to administer effectivity data 6-18
Configure the display of variant expressions 6-20
Control access to configuration data 6-22
Share configurator data 6-22
Performing calculations 6-26
About external calculations 6-26
Calculation phases 6-27
The initiation (PRE) phase example 6-29
The POST processing example 6-30
External calculations process flow 6-30
Setting up external calculations in Product Configurator 6-32
Authenticating calculation Web service requests 6-38
Creating business objects for calculations 6-40
Performing calculations during solve in Product Configurator 6-42
Troubleshooting errors during calculations in Product Configurator 6-44
Enable log files for the external operation block 6-45
Set Product Configuration preferences for Active Workspace 6-46
Run the numeric feature report utility 6-46
Optimize configurator rules to improve solver performance 6-47
Using Product Configurator utilities 6-53
Generate the latest snapshot for modular configurations 6-53
Create advanced search queries for variant rules 6-54
Disable authoring of free form rules 6-54
Enable the Modules tab in Active Workspace 6-54
Allow users to create variant rules or variant criteria 6-55
Allow users to save configuration changes to the current structure or configurator context 6-55

Customizing Product Configurator

About the Product Configurator data model 7-1
Create, modify, update, or delete configurator objects by using recommended ITKs 7-1
Product Configurator business type constants 7-2
Product Configurator business types and the system checks they perform 7-3
Set preferences to manage custom Product Configurator types 7-4
Examples to create a custom package, feature, and inclusion rule 7-5
Allow users to revise custom types by editing the project.xml file 7-7
Create custom objects required for Product Configurator by using BMIDE 7-8
Configure Product Configurator using the Business Modeler IDE 7-8
Product Configurator rules business objects 7-8
Add custom intents for Product Configurator 7-9
Set global uniqueness for a Product Configurator value 7-9
Set context uniqueness for a Product Configurator value 7-10
Define naming rules to customize configurator objects 7-11
Example of naming rules to customize configurator objects 7-12

Create naming rules for configurator objects 7-13
Create naming rules for variant criteria objects 7-17
Customize matrix constraints 7-20
Customize matrix constraint options 7-20
Customize separators in the matrix constraint expression 7-23
Customize formula authoring in Teamcenter configurator 7-25
Customize formula authoring 7-25
Customize the formula handling mechanisms for set expression 7-27
Define expressions using setter APIs 7-29

Create or modify column configuration in Active Workspace

Modify column attributes for the Table view in the Constraints tab 8-1
Modify the header properties in the Grid Editor of the Constraints tab 8-6
Modify columns attributes for views in the Models, Features, or Variants tab 8-9

Using Teamcenter workflows in Product Configurator

Create workflows to release configurator data 9-1
Workflow process example for a configurator rule in a configurator context 9-2
Workflow process example for a configurator feature in a configurator context 9-4
Workflow process example for a configurator feature in a configurator dictionary (global variability) 9-7
Workflow process example for a configurator feature in a configurator dictionary (constrained variability) 9-10
Workflow process example for a specific configurator rule in a dictionary 9-13
Action handlers used in Product Configurator 9-16

1. Setting up Product Configurator

Products are increasingly becoming more complex and customers are demanding greater individual choice. Teamcenter Product Configurator is used to introduce and maintain variability across the product suite at your site. It allows selection of features that are most important to customers. The variant data is independent of any application domain, for example, CAD design or part planning.

As an administrator, you can set up Product Configurator to help marketing owners define the product suite, configuration experts manage variability, and engineers and CAD designers define constraints for using the variability.

Where do I go from here?

Business User	See Product Configurator on Rich Client — Usage or Product Configurator on Active Workspace.
Migration specialist	See Migration of Classic Variants to Product Configurator.
Administrator	
Product Configurator deployment	Product Configurator service components can be **deployed on Linux or Windows machines**. Take a look at the deployment scenarios for understanding this process better.

What is Product Configurator service? How do I configure it?	The **Product Configurator service** is an optional software component used to improve solve performance. As part of this, you can configure the number of requests made by solver nodes, specify the virtual memory limit, and specify log levels by modifying the configuration settings. These tasks are described in the procedures on how to **configure the Product Configurator service**.
Install Product Configurator	You can install Product Configurator **through TEM** or install it **through Deployment Center**.
Administer Product Configurator	To understand the high level tasks for administering product configurator, see the section about **administering Product Configurator**.
Customize Product Configurator	You can use the Business Modeler IDE to create custom objects and optionally define naming rules when users create new objects. These tasks and others are specified as part of the sections on **configuring Product Configurator using the Business Modeler IDE**.

2. Installing Product Configurator (mandatory)

Install Product Configurator using TEM

The following procedures assume that you are installing Product Configurator on an existing Teamcenter set up and that you are familiar with Teamcenter Environment Manager (TEM).

For more information about installing Teamcenter, see *Teamcenter Installation on Windows Using TEM* or *Teamcenter Installation on Linux Using TEM*.

Enterprise tier

Run TEM on the Enterprise tier and select the following features in the **Features** panel:

Feature	Description
Extensions→Product Configurator	Installs the data model and functionality as applicable to Product Configurator.
Base Install→Active Workspace→Server Extensions→Product Configurator	Provides support for working with new options and variants functionality provided by Product Configurator in Active Workspace.
Extensions→Product Configurator Support for Structure Manager	Enables the use of Product Configurator to configure the variability of a product structure.

Client tier

Run TEM on the Client tier and select the following features in the **Features** panel:

Feature	Description
Base Install→Active Workspace→Client→Product Configurator	Provides support for working with new options and variants functionality provided by Product Configurator in Active Workspace.

Install Product Configurator using Deployment Center

Add the Product Configurator application to your existing Teamcenter environment.

Procedure

1. Log on to Deployment Center and select the environment to which you want to add Product Configurator.

2. Go to the **Applications** task. Click **Add or Remove Selected Applications**.

3. In the **Available Applications** panel, use the web browser search to find the following applications.

Application	Description
Product Configurator	Installs the data model and functionality as applicable to Product Configurator.
Product Configurator for Active Workspace	Installs Active Workspace client support for Product Configurator.
Product Configurator Support for Structure Manager	Enables the use of Product Configurator to configure the variability of a product structure.

4. Select the applications, and then click **Update Selected Applications**.

   Deployment Center automatically selects any additional dependent applications.

5. Go to the **Components** task. In the **Selected Components** list, note any remaining components whose configuration status is not **100%**. Select each incomplete component, enter required parameters, and save component settings until all components in the environment show a configuration status of **100%**.

   When all components are fully configured, the **Deploy** task is enabled.

6. Go to the **Deploy** task. Click **Generate Install Scripts** to generate deployment scripts you will use to update affected machines.

   When script generation is complete, note any special instructions in the **Deploy Instructions** panel.

7. Locate deployment scripts, copy each script to its target machine, and then run each script on its target machine.

   For more information about running deployment scripts, see *Deploy task* in *Deployment Center — Usage*.

Delete configuration snapshot data after upgrading or applying patches

The configurator snapshot is the collection of the relevant data required for the constraint solve. It includes the variability, such as families, features, models, and the configurator rules. To optimize the solve time, Teamcenter creates a snapshot of data frozen at a time for a given configuration.

You must delete the configuration snapshot data after upgrading or applying patches. Moreover, if the configurator data is imported from an external system, brief case import, or TC XML import prior

to 12.4 release, you must delete the configuration snapshot data. For releases after 12.4, the system automatically regenerates the configuration snapshot data if the already existing data is out of sync.

Only a user with DBA privileges can delete configuration snapshot data.

To delete configuration snapshot data in rich client:

1. Log on to rich client as user with DBA privileges.
2. In **My Teamcenter**, click **Search** → **Advanced**.
3. From **Select a Search**, choose **General** search.
4. From **Type**, select **Complied Rule Set**, clear all other fields, and click **Search**.
5. Select the configuration snapshot data and click **Delete**.

To delete configuration snapshot data in Active Workspace:

1. Log on to Active Workspace and switch **Group** to **dba**, **Role** to **DBA**, and **Workspace** to **Default**.
2. Click **Advanced Search**, select **General**, and click **Clear All** to clear all other fields.
3. From **Type**, select **Compiled Rule Set** and click **Search**.
4. Select the configuration snapshot data and choose **More Commands** > **Edit** > **Delete**.

Update the fnd0impactingobjects property on SVRs to improve performance after an upgrade

Solution variants have a property called **fnd0ImpactingObjects** for saved variant rules (SVRs). Prior to Teamcenter 13.2 release, the value of this property did not appear on the clients. To enable this property for already created SVRs, it is recommended to run the following utility to improve performance after the upgrade is performed.

add_update_fnd0impactingobjects -u=*TC_USER* **-p=***password* **-g=***dba* **-skip_object_process_limit**

By default this utility only processes all cfg0 objects from 2412 release. However, if you want to process all objects, including non cfg0 objects, use the **-process_non_cfg0_objects** argument while running this utility.

3. Deploying Product Configurator services (optional)

Product Configurator service components

The Product Configurator service components are as follows:

- (Optional) *Microservices and the microservice framework*

  Some Teamcenter applications include microservices as part of their deployment architecture. You can optionally employ a microservice to achieve better performance.

  For more information, see *Teamcenter Installation on Windows Using TEM* or *Teamcenter Installation on Linux Using TEM*.

- (Optional) *Product Configurator service*

  This service improves the solve performance by parallelizing solve requests.

  The microservices framework is a prerequisite for the Product Configurator service.

What is the Product Configurator service?

The Product Configurator service is an optional software component used to achieve better performance by parallelizing solve requests. If it is not installed, all Product Configurator functionalities will continue working as expected. However, adding this component improves solve performance.

The Product Configurator service component consists of three docker containers:

1. **Request Processor**

   Instances of the request processor container act as the API gateway for Product Configurator services. All requests are routed through this gateway that handles the following:

   a. Directing the worker containers to load configurator snapshots.

   b. Distributing solve requests for workers to handle.

2. **Data Grid**

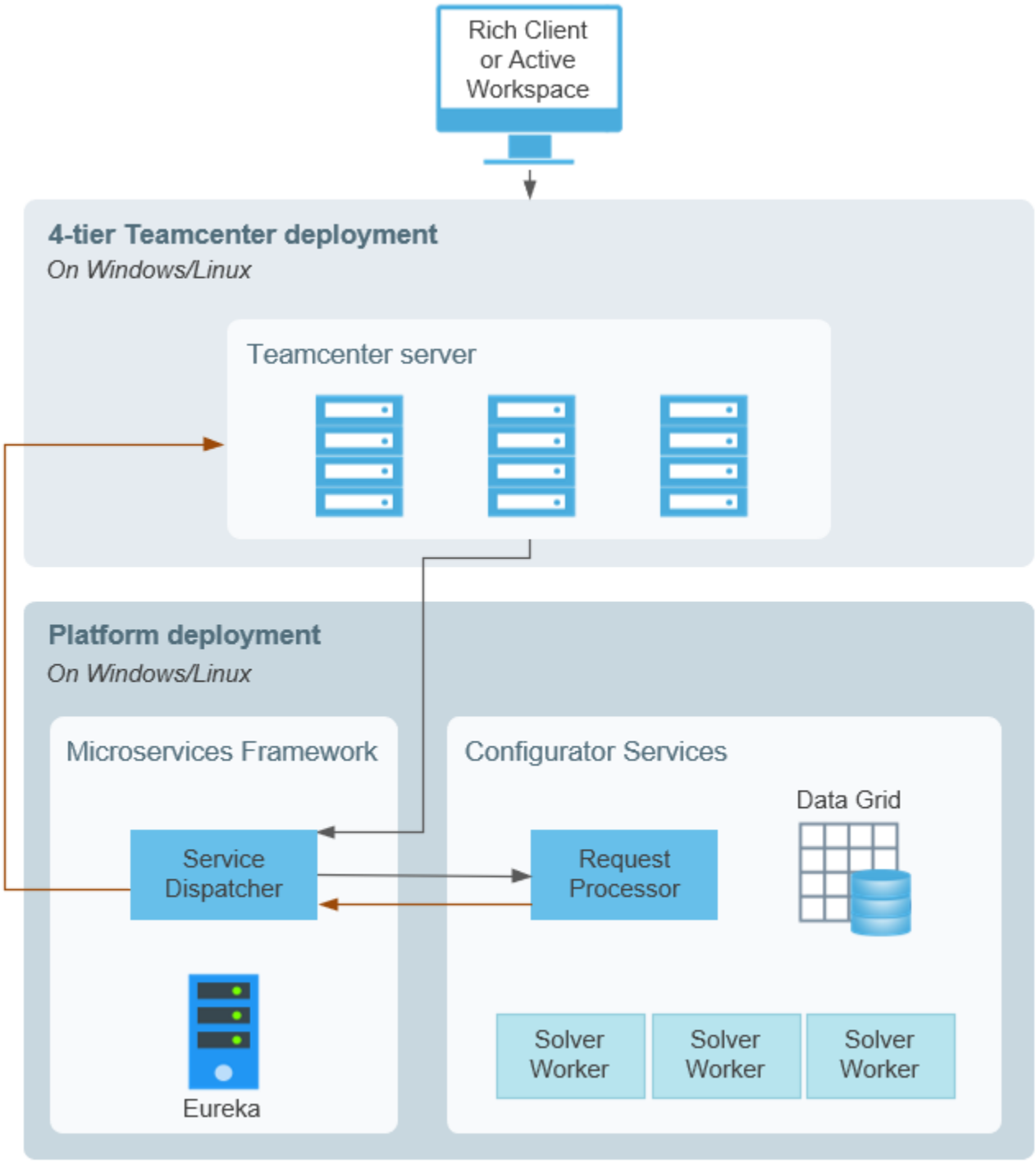
   Instances of the data grid container provide a distributed In Memory Data Grid that is accessible from all containers on the network. It provides a scalable space for coordination and

communication between (potentially) multiple request processor replicas and multiple worker replicas.

3. **Worker**

   Instances of the worker container handle loading the configuration snapshots and calculating the solve request results. Usually, while there are one or two instances of the request processor and data grid containers, there are several workers ready to handle incoming solve requests.

High level configurator service schematic diagram



Deployment scenarios for Product Configurator service components

The following are the deployment scenarios for the Product Configurator service components on Linux and Windows machines.

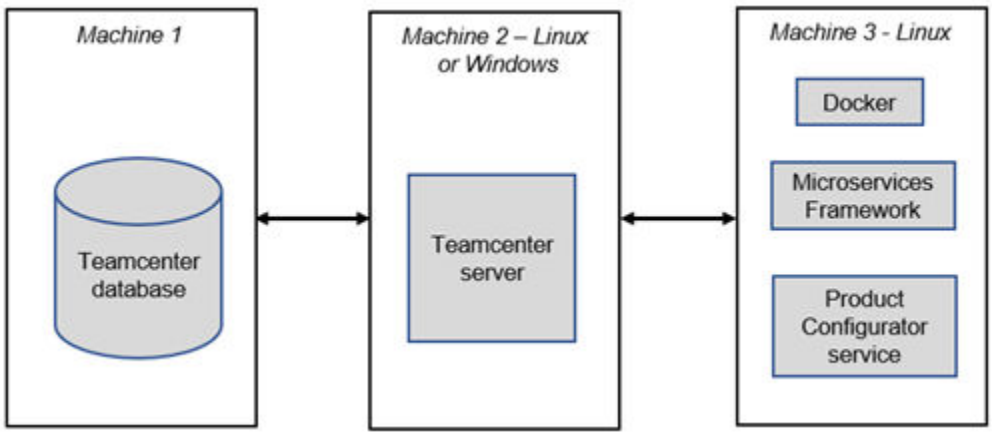
The Product Configurator service is an optional component to improve solve performance by parallelizing solve requests. The microservices framework is a prerequisite for the Product Configurator service.

> Note:
>
> You must not install microservices framework on a Windows machine and the Product Configurator service on a Linux machine or vice versa because this hybrid mode is not supported.

For Linux deployments, the Kubernetes or Docker Swarm container engine can add more instances of the single configured node as needed. For Windows deployments, once a master microservice node is configured, you can add and configure worker microservice nodes in order to increase capacity and provide failover.

For more information, see *Microservices and the microservice framework* in *Teamcenter Installation on Windows Using TEM* or *Teamcenter Installation on Linux Using TEM*.

Product Configurator service components on Linux machines



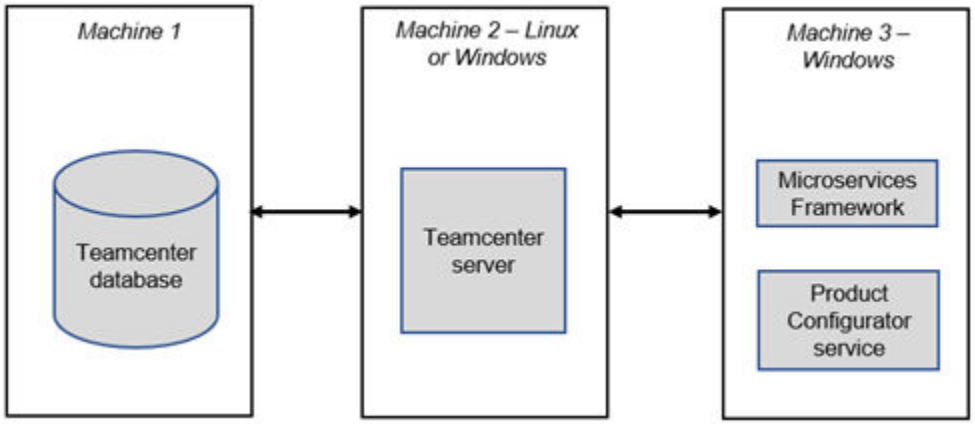
Docker must be installed on a Linux machine on which you can deploy the **Microservices Framework** and the **Product Configurator Service**.

1. Install the microservices framework on a Linux host using Deployment Center or using TEM.

   For more information, see *Install microservices on a Windows host using Deployment Center* in *Teamcenter Installation on Windows Using TEM* or *Install microservices on a Linux host using Deployment Center* in *Teamcenter Installation on Linux Using TEM*.

2. Install the Product Configurator service using Deployment Center OR using TEM.

Product Configurator service components on Windows machines



1. Install the microservices framework on a Windows host using Deployment Center OR using TEM.

   For more information, see *Install microservices on a Windows host using Deployment Center* in *Teamcenter Installation on Windows Using TEM* or *Install microservices on a Linux host using Deployment Center* in *Teamcenter Installation on Linux Using TEM*.

2. Install the Product Configurator service **using Deployment Center** OR **using TEM**.

4. Installing the Product Configurator service (optional)

Task flow for installing the Product Configurator service

The Product Configurator service is an optional component to improve solve performance by parallelizing solve requests. The microservices framework is a prerequisite for the Product Configurator service.

For more information about the deployment scenarios for installing Product Configurator, see **Deployment scenarios for Product Configurator service components**.

1. Install the microservices framework on a Linux host using Deployment Center OR using TEM.

   OR

   Install the microservices framework on a Windows host using Deployment Center OR using TEM.

   For more information, see *Teamcenter Installation on Windows Using TEM* or *Teamcenter Installation on Linux Using TEM*.

2. Install the Product Configurator service **using Deployment Center** OR **using TEM**.

   The process for installing the service on Linux and Windows machines is similar.

3. **Specify the URL and the port to connect Teamcenter to the machine on which microservices are running**.

4. **Specify configurator as the service name to send tasks from Teamcenter to Product Configurator**.

5. **Verify and troubleshoot the Product Configurator service installation**.

Prerequisites for installing the Product Configurator service

> Note:
>
> Before deploying the Product Configurator service, see the Interoperability matrix in the Hardware and Software Certifications knowledge base article on **Support Center** to check compatibility of Product Configurator service and Teamcenter versions.

The versions of the Product Configurator service and Teamcenter must match exactly, for example, Teamcenter 13.1.0.0 is compatible with Product Configurator service 13.1.0.0. For any exception, consult the product owners of the Product Configurator service and get their approval.

For more detailed installation instructions, refer to the generalized information about microservices and the microservice framework, which applies to all the available application microservices, including the Product Configurator service.

For more information, see *Teamcenter Installation on Windows Using TEM* or *Teamcenter Installation on Linux Using TEM*.

The following are the prerequisites for installing the Product Configurator service:

- Docker

  Docker must be installed on a Linux machine on which you can deploy the Product Configurator service as well as the microservices framework component. Hybrid installations where the microservices framework component is installed on Windows and the configurator service is installed on Linux are not supported.

  Note:

  This component is a prerequisite only if the Product Configurator service is deployed as docker containers on Linux. This component is not needed for Windows deployments.

- Microservices framework

  The microservices framework must be installed before deploying the Product Configurator service. This component uses a service registry to maintain a list of running microservice instances and a service dispatcher to receive and route microservice requests.

- Deployment Center or TEM

  If installing through Deployment Center, you must use the latest available Deployment Center to install the Product Configurator service and dependencies. This centralized web application for deploying software to your Teamcenter environments simplifies the process of installing software and automates the deployment.

  If installing through TEM, you must have an understanding of how to install an asynchronous component.

  Currently, Teamcenter can be deployed on Windows and on Linux.

Install the Product Configurator service using TEM

Add the Product Configurator service to your existing Teamcenter environment.

Prerequisites

See *Prerequisites for installing the Product Configurator service*.

Procedure

1. Launch TEM.
2. Select **Product Configurator Service** from the **Microservices** section as a feature to install.

   For more information, see *Microservices and the microservice framework* in *Teamcenter Installation on Windows Using TEM* or *Teamcenter Installation on Linux Using TEM*.
3. Change the number of worker instances as per your configurations.
4. (Linux only) Deploy the docker stack for the framework and the Product Configurator service.

   For more information, see *Deploy the microservice stack* in *Teamcenter Installation on Linux Using TEM*. For the Product Configurator service, deploy the **configurator_services.yml** stack. You must deploy this stack only after deploying the stack for the microservice framework.

   > Note:
   >
   > The Configurator Service distributes the system load across all worker containers. Therefore, the more you can deploy, the better is the response time. However, do not deploy more worker containers than there are **hardware thread contexts**.
5. Copy the **signer_config** folder from the microservice installation folder to the Teamcenter installation *TC_DATA* folder for the Product Configurator service.

Install the Product Configurator service using Deployment Center

Add the Product Configurator service to your existing Teamcenter environment.

Prerequisites

See *Prerequisites for installing the Product Configurator service*.

Procedure

1. Log on to Deployment Center and select the environment to which you want to add the Product Configurator service.
2. Go to the **Applications** task. Click **Add or Remove Selected Applications**.

3. In the **Available Applications** panel, use the web browser search to find **Product Configurator Service**. Select the service, and then click **Update Selected Applications**.

   Deployment Center automatically selects any additional dependent applications.

4. In the **Components** task, provide standard configuration parameters.

   For more information, see *Install microservices on a Windows host using Deployment Center* in *Teamcenter Installation on Windows Using TEM* or *Install microservices on a Linux host using Deployment Center* in *Teamcenter Installation on Linux Using TEM*.

5. Change the number of worker instances as per your configurations.

   In the **Selected Components** list, select **Microservice Node** and specify the worker instances in **Services** > **Product Configurator Service**.

6. When you finish entering values, click **Save Component Settings**.

7. In the **Selected Components** list, note any remaining components whose configuration status is not **100%**. Select each incomplete component, enter required parameters, and save component settings until all components in the environment show a configuration status of **100%**.

   When all components are fully configured, the **Deploy** task is enabled.

8. Go to the **Deploy** task. Click **Generate Install Scripts** to generate deployment scripts you will use to update affected machines.

   When script generation is complete, note any special instructions in the **Deploy Instructions** panel.

9. Locate deployment scripts, copy each script to its target machine, and then run each script on its target machine.

   For more information about running deployment scripts, see *Run the deployment scripts* in *Deployment Center — Usage*.

10. (Linux only) Deploy the docker stack for the framework and the Product Configurator service.

    For more information, see *Deploy the microservice stack* in *Teamcenter Installation on Linux Using TEM*. For the Product Configurator service, deploy the **configurator_services.yml** stack. You must deploy this stack only after deploying the stack for the microservice framework.

> Note:
>
> The Configurator Service distributes the system load across all worker containers. Therefore, the more you can deploy, the better is the response time. However, do not deploy more worker containers than there are **hardware thread contexts**.

11. Copy the **signer_config** folder from the microservice installation folder to the Teamcenter installation *TC_DATA* folder for the Product Configurator service.

Specify the URL and the port to connect Teamcenter to the machine on which microservices are running

Set the **TC_Microservices_Base_URL** site preference to define how Teamcenter connects to the machine on which microservices are running.

> Example:
>
> If the service dispatcher is deployed at *tcServices.company.com* on port *9090*, then specify **http://tcServices.company.com:9090**.
>
> Use **https** if the service dispatcher supports **https** communication, for example, **https://tcServices.company.com:9090**. Do not specify a slash at the end of URL string.

For information about retrieving a list of preferences, see *Where can I get a list of preferences?* in *Teamcenter Preferences*.

Specify configurator as the service name to send tasks from Teamcenter to Product Configurator

Set the **TC_Microservices_Installed_Services** site preference to specify the list of installed services. By default, it is set to **configurator** for Product Configurator.

Specifies the list of installed services (for example, **configurator**). Teamcenter needs to know whether a service is deployed in order to decide whether or not to send tasks to the service. If the service name appears in this preference the tasks are sent to the machine deployed to the Teamcenter Service environment, that is, the machine on which microservices are installed. Otherwise, these tasks are performed in the Teamcenter process.

For information about retrieving a list of preferences, see *Where can I get a list of preferences?* in *Teamcenter Preferences*.

Verify and troubleshoot the Product Configurator service installation

Verify the installation

- The deployment can be verified using the following URL:

  http://tcServices.company.com:9090/configurator/ping

  This URL displays the configurator services build ID and the Teamcenter build ID.

  > Note:
  >
  > The end point of this URL and its content are subject to change in future releases.

- A detailed health check can be achieved using any docker container monitoring tool such as Portainer.

Troubleshoot the installation

- Turn on curl logging for detailed connection diagnostics.
  - This requires Access Manager Bypass because sensitive Information might get logged.
  - In Teamcenter: `$TC_DATA/logger.properties file logging.logger.Teamcenter.MicroServicesUtils=DEBUG`
- Run the worker at log level DEBUG.
  - Update the *solverworker.env* docker compose file that deploys the worker.

    ```
    TC_LOGGER_CONFIGURATION=/tc/data/debug
    ```

    > Note:
    >
    > Perform this step for debugging only if you face issues while the Product Configurator service is starting.

- Enable debugging by using Microservice Parameter Store command line utility.

  For more information, see Microservice Parameter Store command line utility documentation for options. The service name used in options is **configurator** for Product Configurator service.

  Example command: **log-level set configurator Debug**

5. Configuring the Product Configurator service

Modify mandatory and optional preferences

1. Log on to Teamcenter as a Teamcenter administrator to modify the following mandatory preferences:

   - **TC_Microservices_Base_URL**

     Specify the microservices framework URL.

     During installation, you deployed the docker stack in which all service containers run. Only the **service_dispatcher** service exposes a port that is accessible from outside the stack. You can see this port using the **docker service ls** command.

     This preference defines how Teamcenter can connect to the microservices. For example, if you deployed the service dispatcher at **tcServices.company.com** on port **9090**, then you should specify **http://tcServices.company.com:9090**.

     > Caution:
     >
     > It is important *not* to have the trailing forward slash at the end of the preference value. For example, entering **http://tcServices.company.com:9090/** is incorrect.

   - **TC_Microservices_Installed_Services**

     Add **configurator** to the list.

     The service dispatcher you deployed routes incoming requests to a container that provides the requested service. The service dispatcher recognizes Product Configurator service requests based on the first element in the URL path that follows the **TC_Microservices_Base_URL** you specified above.

     http://tcServices.company.com:9090/configurator/... is recognized as a Product Configurator service request.

     http://tcServices.company.com:9090/ProductConfigurator/... is not recognized as a Product Configurator service request. It is assumed to be a request to another service because the first part of the path does not match the exact string **configurator**.

     Teamcenter decides whether or not to send tasks to the service based on whether a Product Configurator service is deployed. If the Product Configurator service appears in this preference,

some tasks are sent to the Product Configurator service you previously deployed. Otherwise, these tasks are performed in the Teamcenter process.

2. Modify the following optional preferences:

   - **Cfg0_Service_Connections_Count**

     This preference specifies the number of parallel connections per user.

     Setting this number to a value that is higher than the actual number of containers in the solver worker instanced in the Product Configurator service causes a serialization of requests without the advantage of parallel processing.

     If there is only a single user in the system, set this number to match the nominal container count. In practice, your containers will be shared across many users. Therefore, it is expected that you deploy more containers than a single user should connect to in parallel.

     If you do not have information on this, set this value to 16.

     Depending on the size of a task, Teamcenter may decide to open fewer parallel connections but not more.

     Too many parallel connections from too many users can overload the system. The maximum value allowed is 128.

   - **Cfg0_Content_Solve_Service_Batch_Size**

     This preference specifies the target size of a content solve request. This number must not be undershot unless the task is relatively small, compared to the many variant conditions to solve.

     Teamcenter queues the requests according to the target request size. It breaks up the tasks into multiple requests, each having the target size, when appropriate. These requests can be processed in parallel according to the **Cfg0_Service_Connections_Count** preference.

   - **Cfg0_ExpandExpression_Service_Batch_Size**

     This preference specifies the target size of an **ExpandExpression** request. This value must not be undershot unless the task is relatively small, compared to the many families to expand.

     Teamcenter queues requests according to the target request size. It breaks up the tasks into multiple requests, each having the target size, when appropriate. These requests can be processed in parallel according to the **Cfg0_Service_Connections_Count** preference.

   - **Cfg0_GetValidValues_Service_Batch_Size**

     This preference specifies the target size of a **GetValidValues** request. This value must not be undershot unless the task is relatively small, compared to the many values to validate.

Teamcenter queues requests according to the target request size. It breaks up the tasks into multiple requests, each having the target size, when appropriate. These requests can be processed in parallel according to the **Cfg0_Service_Connections_Count** preference.

> Note:
>
> Higher **Min Batch Sizes** reduce the number of parallel requests for small problems:
>
> "# Parallel Requests" <= "Problem Size" / "Min Batch Size"
>
> Lower **Min Batch Sizes** increase the degree of parallelism until the maximum value is reached.

Configure the Product Configurator service

1. Before deploying the Product Configurator service, review the *configurator_services.xml* file generated in the installation folder.

   > Note:
   >
   > Any change in this file requires re-deployment of the Product Configurator service stack.

2. Configure the solver worker by modifying the following configuration settings.

   - **TC_LOGGER_CONFIGURATION**

     Change the log level in the solver worker container. This is used to run a solver worker in debug mode in order to understand the root cause of an issue.

     > Warning:
     >
     > Logging at the DEBUG level may log sensitive information such as an FMS file ticket of a configurator snapshot.

     - *Linux*: Modify the **solverworker.env** file that is generated in the **install** folder.
     - *Windows*: Modify the **logger.properties** file in the *TC_ROOT***microservices\worker-***Product Configurator Service Version***wntx64\data** directory.

       > Note:
       >
       > The log level changed by this setting is applied only when the solver worker is starting. To update the log level after it is started, modify the log level of the Product Configurator service by using Microservices Parameter Store.

For more information and options, see Microservice Parameter Store command line utility documentation. The service name used in options is **configurator** for Product Configurator service.

Example command: **log-level set configurator Debug**

- **RequestLimit**

  Sets the limit on the number of connections that an individual worker container handles during its life. When a worker reaches capacity, it exits, and a new worker automatically takes over. If **RequestLimit** is set to 0, the worker never expires. Recycling containers can help maintain a responsive system for long service uptimes. The value must be a positive integer.

  - *Linux*: Modify the **solverworker.env** file that is generated in the **install** folder.
  - *Windows*: Modify the **solverworker.json** file in the *TC_ROOT***microservices\services_config** directory by adding a manual entry in the **environment** section.

    Example:

    ```
    "environment":[
          .. .. .. ..
          “RequestLimit = 50000”
        ]
    ```

- **VirtualMemoryLimit**

  Sets the limit on the amount of virtual memory with which a worker container can continue after serving a request. When a worker reaches capacity, it exits, and a new worker automatically takes over. If **VirtualMemoryLimit** is set to 0, the worker never expires. Recycling containers can help maintain a responsive system for long service uptimes. The value should be a positive integer that specifies the amount of virtual memory the worker process can allocate in megabytes (MB). Most container management applications, including Docker, monitor and track virtual memory that is exclusively reserved and committed to a specific container. This is usually substantially less than the virtual memory the container has allocated (but not yet used, or not used exclusively). The actual amount of virtual memory in use for a given container is **controlled (and potentially limited)** by the container management application.

  - *Linux*: Modify the **solverworker.env** file that is generated in the **install** folder.
  - *Windows*: Modify the **solverworker.json** file in the *TC_ROOT***microservices\services_config** directory by adding a manual entry in the **environment** section.

    Example:

    ```
    "environment":[
          .. .. .. ..
    ```

```
    “VirtualMemoryLimit = 32000”
]
```

3. Configure the request processor by modifying the following settings:

   - **LOG_LEVEL**

     Enables verboseness of application logging in the request processor.

     - *Linux*: Modify the **solverrequestprocessor.env** file generated in the **install** folder.

     - *Windows*: Modify the **solverrequestprocessor.json** file in the *TC_ROOT***microservices\services_config** file by adding a manual entry in the **environment** section.

       Example:

       ```
       "environment":[
             .. .. .. ..
             “LOG_LEVEL = DEBUG”
           ]
       ```

     > Note:
     >
     > The log level changed by this setting is applied only when the request processor is starting. To update the log level after it is started, modify the log level of the Product Configurator service by using Microservices Parameter Store.

   - **ResponseRetention__RetentionPeriod**

     Specifies the lifetime retention period of response results. If this value is set to a non-zero, positive integer, pre-calculated results are returned, as long as the configurator snapshot ID and request body are unchanged. These results are stored for the duration specified in this parameter from the time it is last accessed. The values specified are in seconds. Set this value to **0** if no pre-calculated results are needed.

     A large retention period increases the memory footprint of the data grid container but leads to a higher hit ratio.

     - *Linux*: Modify the **solverrequestprocessor.env** file generated in the **install** folder.

     - *Windows*: Modify the **solverrequestprocessor.json** file in the *TC_ROOT***microservices\services_config** file by adding a manual entry in the **environment** section.

       Example:

```
"environment":[
      .. .. .. ..
      “ResponseRetention__RetentionPeriod = 3600”
    ]
```

- **SharedState__TopicIdleLimitInMinutes**

  Specifies how long before a configurator snapshot must be inactive before its assigned workers can be released for other work.

  When a request needs a new configurator snapshot, the system assigns some worker containers to the new configurator snapshot. If no workers are free, some are allocated from existing configurator snapshots.

  > Note:
  >
  > You can minimize the number of active configurator snapshots by using a common rule date.

  - *Linux*: Modify the **solverrequestprocessor.env** file generated in the **install** folder.

  - *Windows*: Modify the **solverrequestprocessor.json** file in the *TC_ROOT***microservices\services_config** file by adding a manual entry in the **environment** section.

    Example:

    ```
    "environment":[
          .. .. .. ..
      “SharedState__TopicIdleLimitInMinutes = 60”
      ]
    ```

- **EventBus__WorkTimeoutInSeconds**

  Specifies the time to wait for a worker response (you may need to increase this if workers are overloaded).

  The system uses a queue for every active, unique configurator snapshot. The request processor transfers requests to the queue that corresponds to the configurator snapshot of the request.

  If a request stays in the queue for a long time, a timeout occurs and a **504 Gateway Timeout Status** is returned to Teamcenter. This protects the system from overheating. In case this error is returned to Teamcenter, it falls back to in-process processing resulting in a performance lag for end users.

  - *Linux*: Modify the **solverrequestprocessor.env** file generated in the **install** folder.

- *Windows*: Modify the **solverrequestprocessor.json** file in the *TC_ROOT***microservices\services_config** file by adding a manual entry in the **environment** section.

  Example:

  ```
  "environment":[
        .. .. .. ..
    “EventBus__WorkTimeoutInSeconds = 30”
    ]
  ```

- **EventBus__MaxTopicQueueDepth**

  The system uses a queue for every active, unique configurator snapshot. The request processor transfers requests to the queue that corresponds to the configurator snapshot of the request.

  If the queue is very long, incoming requests are rejected and a **503 Service Unavailable** status is returned to Teamcenter. This protects the system from overheating. In such a case, the system falls back to in-process processing, resulting in a performance lag for end users.

  - *Linux*: Modify the **solverrequestprocessor.env** file generated in the **install** folder.

  - *Windows*: Modify the **solverrequestprocessor.json** file in the *TC_ROOT***microservices\services_config** file by adding a manual entry in the **environment** section.

    Example:

    ```
    "environment":[
          .. .. .. ..
      “EventBus__MaxTopicQueueDepth = 0”
      ]
    ```

View advanced information about the Product Configurator service including logs and requests

You can access the Product Configurator service by using the *http://<host>:<port>/configurator/status* URL, for example, *http:// tcServices.company.com:9090/configurator/status*.

> Note:
>
> The URI format and its content are subject to change in future releases.

- By default, this URL provides data for the current day (since the start of the day) and that generated in the time span of 1 hour.

- You can fine tune this by providing URL query parameters in days and the by specifying the time span.

  Example:

  http://<host>:<port>/configurator/status?days=10&span=15

  This provides details for the last 10 days in the time span of 15 minutes.

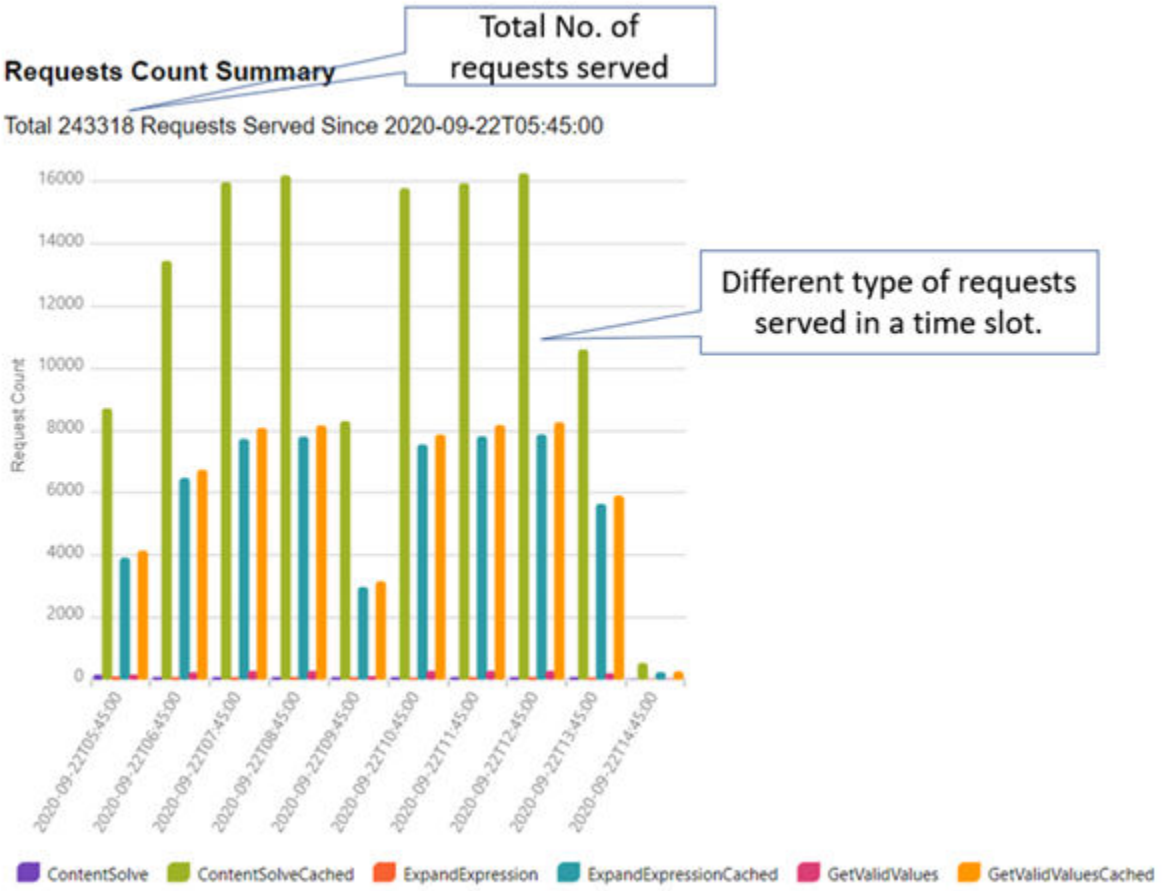
  You must specify **days** as a complete integer and **span** in multiples of 5.

  The details are not persisted. These are stored in the data grid component and are accessible if the data grid is running. This data is cleared when the data grid is restarted.

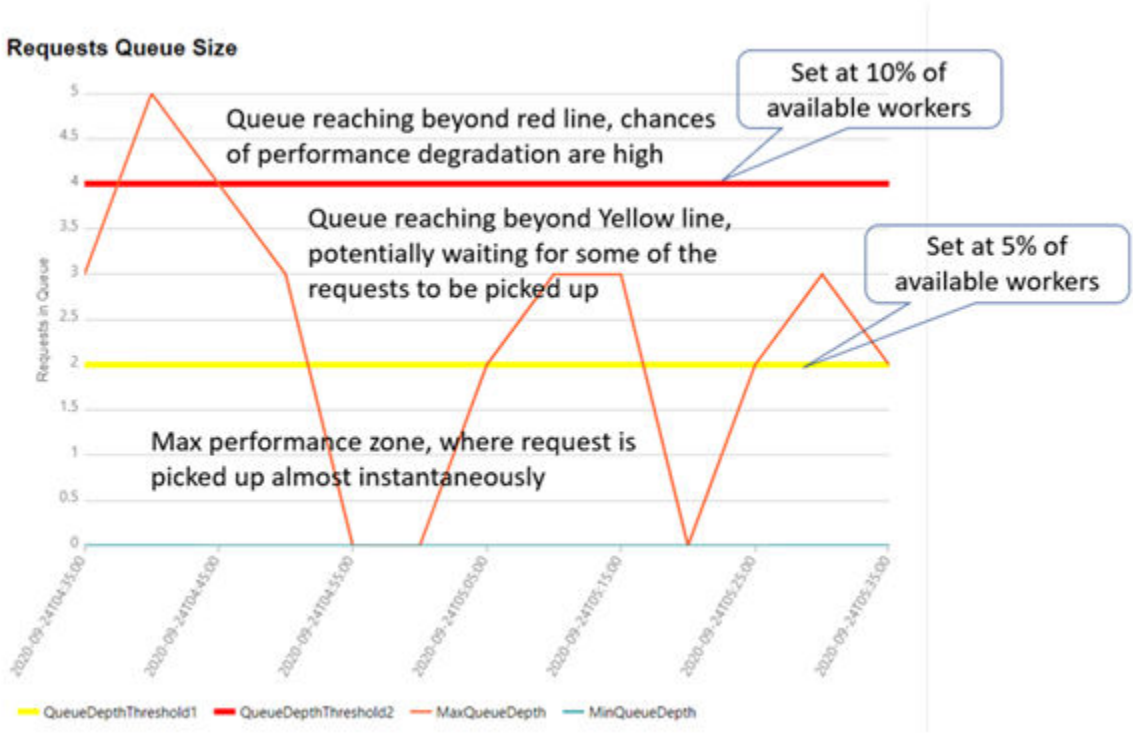
This URL provides the following details:

- Product Configurator service version and build ID
- Teamcenter build ID
- Most common environment settings
- Log level
- The number of data grids, request processors, and solver workers, including the uptime
- A graph about requests served and requests in the queue for solver workers

The following chart represents the total requests served since the given time. The count is a total of all **requests that are served from the http cache in the data grid** and processed by the solver worker.



The following chart represents the request queue pattern over a period of time. The depth is the sum of maximum queue depth for a given time interval across all configurator snapshots being solved.



All about Docker for the Product Configurator service

The Product Configurator service runs within Docker containers. Before deploying the Product Configurator service, you must install Docker on a Linux machine. Refer to Docker installation instructions at **https://docs.docker.com**.

In addition to the Docker installation instructions, refer to:

- Docker post-install instructions to configure Docker to restart on system boot.
- The *Teamcenter Software Certifications* section of the hardware and software certifications knowledge base article on **Support Center** for certified versions of SUSE Linux and Red Hat Linux, as well as Docker software.
- For more information about installing and configuring Docker, working with Docker containers, and Docker troubleshooting, see Deploy Docker for microservices on Linux hosts in *Teamcenter Installation on Linux Using TEM*.

Two Docker containers manage Product Configurator service deployment

Product Configurator service deployment involves two Docker containers with the following roles:

Service Dispatcher	Forwards requests to a Product Configurator service container and queries the Eureka Server to determine the location of the Product Configurator service.
Eureka Server (service registry)	Processes the registration and heartbeat messages sent by the Product Configurator service.

6. Administering Product Configurator

About administering Product Configurator

Task	Description
Specify the validation mode for the Variant Configuration view	Set the **Cfg0DefaultValidationMode** site preference to specify the default mode used to validate the configuration expression in the **Variant Configuration** view. By default, it is set to **Order** mode.
Allow or disallow users from creating and editing variant option data	You can allow or disallow users from creating and editing variant option data by setting the **PCA_enable_authoring** site preference. By default, it is set to **TRUE**.
Administering validation mode in Active Workspace using preferences	Set preferences for validation modes in Active Workspace for **Overlay** or **Order** mode.
Set business object constants in Business Modeler IDE	Set business object constants to define the type of expression objects and to specify whether the configurator objects should be included in **Where Used** searches.
Configure Name instead of ID as the primary business attribute	Configure corresponding views to display either **ID** or **Name**, based on the configuration setting. The solve and internal processing of configurator expressions are performed using an ID only.
Configure the display format of variant expressions	Configure the display of variant expressions by using site preferences and by specifying expression formats.
Control access to configuration data	Control who has access to create and allocate objects to the group, family, and features in your dictionary
Share configurator data	Share configurator data by using Multi-Site Collaboration, **Briefcase** data transfer with the high level TC XML format to transfer objects offline, or TC XML low level commands to transfer objects offline.
Create workflows to release configurator data	Create workflows to release configurator data by using specific workflow handlers. This ensures that when the configurator features are released, their families and configurator allocations are automatically included in the same workflow process.

Specify the validation mode for the Variant Configuration view

You can set the **Cfg0DefaultValidationMode** site preference to specify the default mode used to validate the configuration expression in the **Variant Configuration** view. By default, it is set to **Order** mode.

For information about retrieving a list of preferences, see *Where can I get a list of preferences?* in *Teamcenter Preferences*.

Allow or disallow users from creating and editing variant option data

You can allow or disallow users from creating and editing variant option data by setting the **PCA_enable_authoring** site preference. By default, it is set to **TRUE**.

For information about retrieving a list of preferences, see *Where can I get a list of preferences?* in *Teamcenter Preferences*.

Administering validation mode in Active Workspace using preferences

Active Workspace supports two types of validation modes in the **Settings** panel of the **Variant Configuration** tab, **Order** and **Overlay** mode. The default profile is based on the **PCA_Default_Solve_Mode** preference. This preference is not available by default.

As an administrator, you must create the **PCA_Default_Solve_Mode** preference and you can set it to **pca0Order** for **Order** (default) or **pca0Overlay** for **Overlay** mode.

After you create the preference, users can use one of the modes as follows:

- **Order** (default profile)

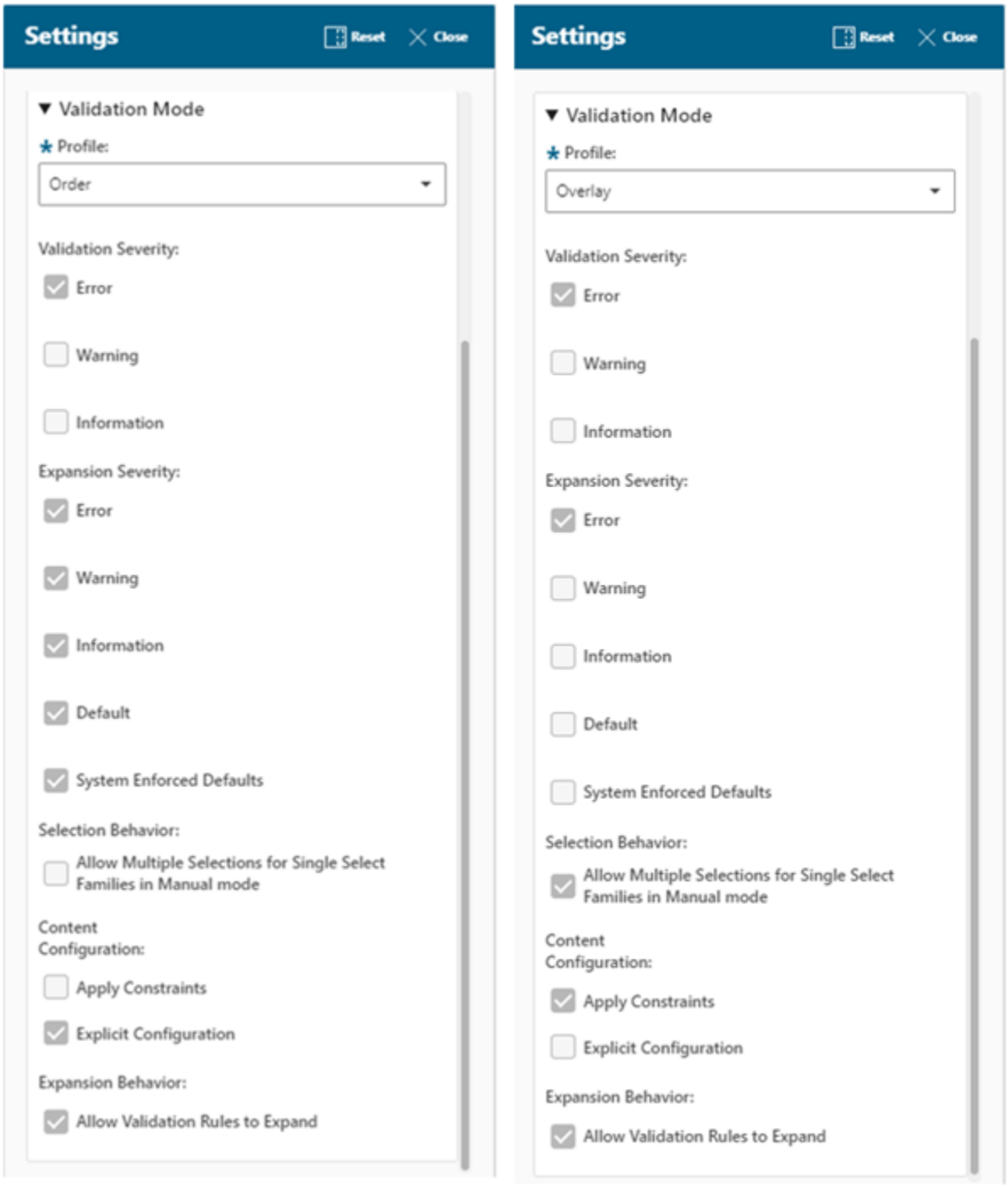
  This mode is intended for users who want to quickly arrive at a 100% BOM. A 100% BOM is used to create a prototype, perform simulations, compute the price, calculate the weight of the product, or visualize the structure.

  The system uses only user selections or the previously expanded order when the configuration is applied to the content. It is the users' responsibility to use either the guided configuration or the manual configuration and expand the order features.

- **Overlay**

  This mode does not expect users to specify the complete configuration. It is intended for developing a new product or making changes to an existing product.

The system allows users to overlay multiple product configurations and multiple variations of the same product.



Include a customized validation mode

In addition to the **Overlay** and **Order** modes, as an administrator, you can include a custom validation mode by creating the **PCA_solver_profiles** site preference. This preference is not available by default. Make sure that there are no extra spaces when you create this preference value.

Default validation mode example:

In the following example, the first four parameters are mandatory and the last two are optional for the **pca0Overlay** and **pca0Order** validation modes.

```
{
        "solverProfiles" : [
            {
                "pca0ProfileName": "pca0Order",
                "pca0ValidationSeverity": "Error",
                "pca0ExpansionSeverity": "System Enforced Defaults",
                "pca0AllowMultipleSelections": "false",
                "pca0EnableExplicitContentConfiguration": "true",
                "pca0ApplyConstraints": "false",
                "pca0AllowValidationRulesToExpand": "true"
            },
            {
                "pca0ProfileName": "pca0Overlay",
                "pca0ValidationSeverity": "Error",
                "pca0ExpansionSeverity": "Error",
                "pca0AllowMultipleSelections": "true",
                "pca0EnableExplicitContentConfiguration": "false",
                "pca0ApplyConstraints": "true",
                "pca0AllowValidationRulesToExpand": "true"
            }
        ]
    }
```

Custom validation mode example after adding **My Order Mode**:

In the following example, the first four parameters are mandatory and the last two are optional for the **pca0Overlay**, **pca0Order**, **My Order Mode** validation modes.

```
{
        "solverProfiles" : [
            {
                "pca0ProfileName": "pca0Order",
                "pca0ValidationSeverity": "Error",
                "pca0ExpansionSeverity": "System Enforced Defaults",
                "pca0AllowMultipleSelections": "false",
                "pca0EnableExplicitContentConfiguration": "true",
                "pca0ApplyConstraints": "false",
```

```
                "pca0AllowValidationRulesToExpand": "true"
            },
            {
                "pca0ProfileName": "pca0Overlay",
                "pca0ValidationSeverity": "Error",
                "pca0ExpansionSeverity": "Error",
                "pca0AllowMultipleSelections": "true",
                "pca0EnableExplicitContentConfiguration": "false",
                "pca0ApplyConstraints": "true",
                "pca0AllowValidationRulesToExpand": "true"
            },
            {
                "pca0ProfileName":"My Order Mode",
                "pca0ValidationSeverity":"pca0Error",
                "pca0ExpansionSeverity":"pca0Default",
                "pca0AllowMultipleSelections":"true",
                "pca0EnableExplicitContentConfiguration": "true",
                "pca0ApplyConstraints":"true",
                "pca0AllowValidationRulesToExpand":"false"
        }
        ]
    }
```

In this example, **pca0ProfileName** is set to **My Order Mode** and this is the name that appears in the **Settings** panel of the **Variant Configuration** tab.

Field	Value
Name	**PCA_solver_profiles**
Protection Scope	**Site**
Category	**Active Workspace**
Type	**String**
Multiple	**Single**
Description	As appropriate
Value	Example as above

- *Preference validation*

  The validation process analyzes the structure of the preference value. The profiles are stored as listed in the preference.

- *Duplicates*

  No duplicate check is implemented currently for either profile names or profile parameters.

- *Validation of parameters*

The parameters are parsed and saved into profiles by name, value, and type. The parameters type definition happens in the service initialization process. If the preference being parsed contains a parameter that was not defined, this parameter is saved in the default **string** format.

- *Mandatory profiles*

  Currently, there is no check on mandatory profiles.

  The default **Overlay** or **Order** profile names must be provided in the preference as internal keys to be localized on the client:

  - Order: **pca0Order**
  - Overlay: **pca0Overlay**

  > Note:
  >
  > The **Custom** keyword is reserved for a profile name. You must not define a profile with the name **Custom**. It is assigned from the server while loading a saved variant rule (SVR) in case the SVR profile does not match any of the existing profiles in the preference. Currently, there is no check for **Custom** as a reserved word in the list of profiles from the preference.

- *Mandatory and optional parameters*

  The validation process is flexible. Any parameter from the parsed preference is saved to the profile. Only a few mandatory parameters are defined. If they are not provided, the validation process fails:

Parameter type	Profile name	Value	Allowed values
Mandatory	**Pca0ProfileName**	String	
Mandatory	**Pca0ValidationSeverity**	String	**pca0Error**, **pca0Warning**, and **pca0Information**
Mandatory	**Pca0ExpansionSeverity**	String	**pca0Error**, **pca0Warning**, **pca0Information**, **pca0Default**, and **pca0SysDefault**
Mandatory	**pca0EnableExplicitContent-Configuration**	String	**true** or **false**
Optional	**Pca0AllowMultipleSelections**	String	**true** or **false** If the value is not provided, the default value is considered as **true**.
Optional	**Pca0ApplyConstraints**	String	**true** or **false**
Optional	**Pca0AllowValidationRulesToExpand**	String	**true** or **false**

- *Severity formats*

  The following are the allowed severity formats:

  - Error: **“pca0Error”**
  - Warning: **“pca0Warning”**
  - Information: **“pca0Information”**
  - Default: **“pca0Default”**
  - Soft Defaults: **“pca0SysDefault”**

  The following combination is not supported for validation and expansion severities:

  - **pca0ValidationSeverity: ‘pca0Error’**
  - **pca0ExpansionSeverity: ‘pca0Warning’**

  The expansion severity cannot be higher than the validation severity. For example, the following is an invalid combination:

  - **pca0ValidationSeverity: ‘pca0Information’**
  - **pca0ExpansionSeverity: ‘pca0Error’**

Set the customized validation mode as the default validation mode

You can set the customized **My Order Mode** validation mode as the default validation mode by creating the **PCA_Default_Solve_Mode** site preference. This preference is not available by default and is intended to replace the **Cfg0DefaultValidationMode** preference that allows **Order** or **Overlay** mode. After setting this preference, the **Variant Configuration** view in Active Workspace uses **My Order Mode** as the default profile.

Field	Value
Name	**PCA_Default_Solve_Mode**
Protection Scope	**Site**
Category	**Active Workspace**
Type	**String**
Multiple	**Single**
Description	As appropriate
Value	Name of the custom profile you created, for example, **My Order Mode**

Set business object constants

You can set business object constants to define the type of expression objects and specify whether configurator objects should be included in **Where Used** searches.

You should set the following business object constants in the Business Modeler IDE:

Business object constant	Defines	Default value
Fnd0CFilterVariantExpressionType	The type of expression objects that objects which can be configured by variants use to store their variant conditions.	**Fnd0VariantExpression**
Fnd0WhereUsed	Whether configurator objects are found by "where used" searches.	**true**
Cfg0DefaultValueType	The type of default feature used by family objects.	None
Cfg0ThreadType	The type of thread used by instantiable revision objects types (a subtype of the specified thread type may also be used). Abstract classes should not define a value for this business type constant.	Empty

Using name or ID as the primary business attribute

Use Name or ID as the primary business attribute

Features undergo change of displayable properties throughout stages of planning and design. Specific features, their purpose, and user-facing naming conventions for features and families may change as the product develops. However, you cannot change IDs for your configurator objects. Moreover, IDs may be hidden from users based on your local business process.

Teamcenter allows you to use **Name** or **ID** as a primary business-relevant attribute in Product Configurator, based on a global setting. You can view and edit a name for a feature, a family, or a group across all revisions, per your business requirement, in all Product Configurator dialog boxes and views, such as **Variability Explorer**, **Configurator Rules**, **Variant Expression Editor**, **Availability Matrix**, **Saved Variant Rules**, and **Variant Configuration**.

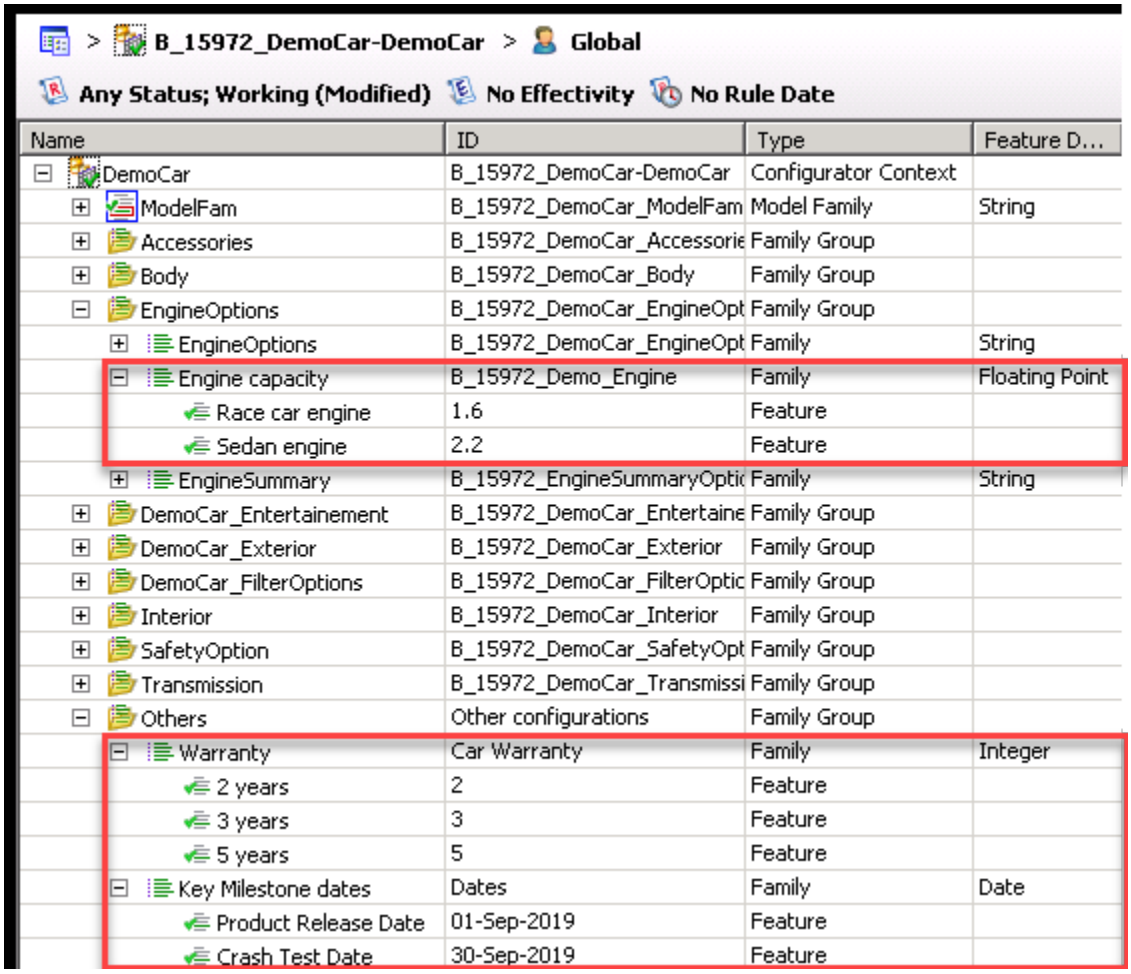
> Note:
>
> The system cannot be configured to use both **Name** and **ID** as the primary attribute at the same time. Additionally, you cannot use **Name** as a primary attribute for Product Configurator *allocation* objects.

You can configure corresponding views to display either **ID** or **Name**, based on the configuration setting. The solve and internal processing of configurator expressions are performed using an ID only.

A **Name** property of configurator object can be modified, but a corresponding ID of configurator objects remain unchanged.

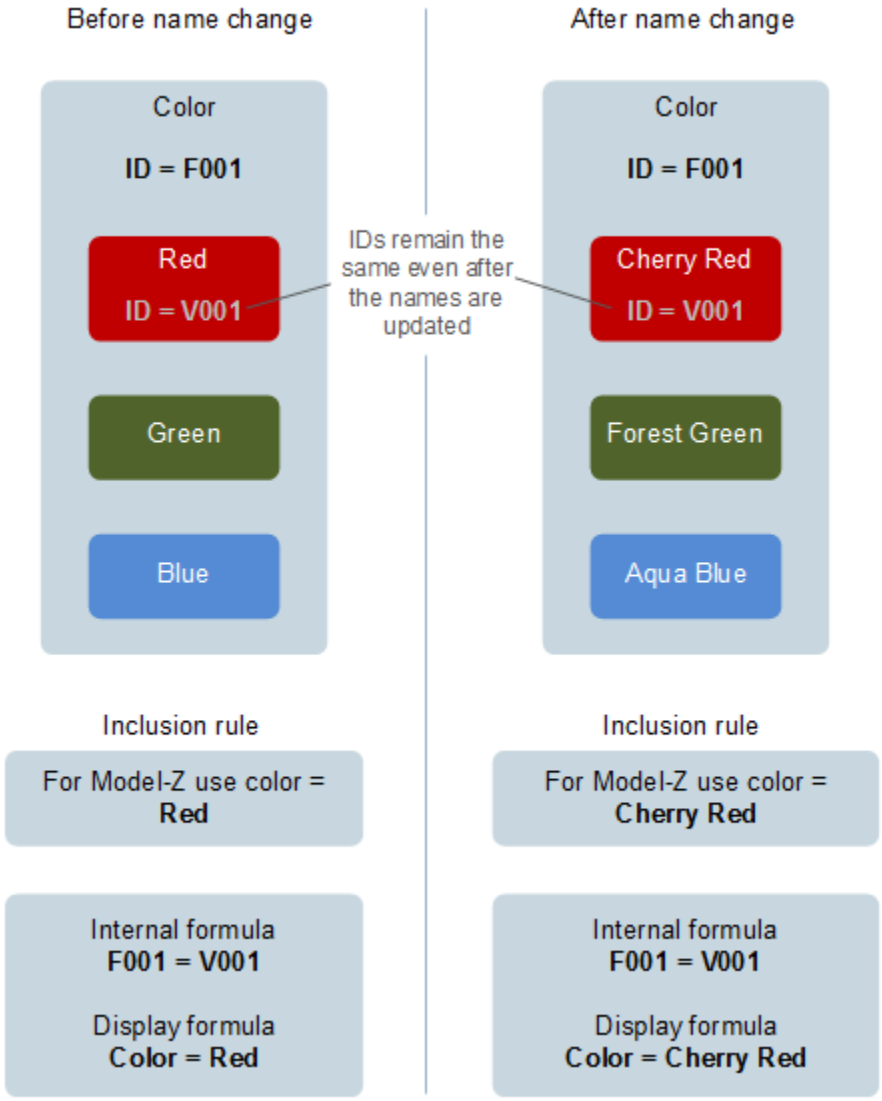
You can deploy the system with the **Cfg0PrimaryBusinessRelevantAttribute** global constant that allows you to use either **Name** or **ID** as the primary business-relevant attribute. When the system is configured with the global constant that points to **Name** as a primary business property, all Product Configurator views display **Name** instead of **ID** in the corresponding areas.

Valid values for the **Cfg0PrimaryBusinessRelevantAttribute** global constant are **Cfg0AbsConfiguratorWSO.cfg0ObjectId** and **Cfg0AbsConfiguratorWSO.object_name**. **Cfg0AbsConfiguratorWSO.cfg0Object_name** is the default value. Any value other than these two values, an empty value, or an absence of this global constant is treated as the default value.



Name	ID	Type	Feature D...
DemoCar	B_15972_DemoCar-DemoCar	Configurator Context	
ModelFam	B_15972_DemoCar_ModelFam	Model Family	String
Accessories	B_15972_DemoCar_Accessorie	Family Group	
Body	B_15972_DemoCar_Body	Family Group	
EngineOptions	B_15972_DemoCar_EngineOpt	Family Group	
EngineOptions	B_15972_DemoCar_EngineOpt	Family	String
Engine capacity	B_15972_Demo_Engine	Family	Floating Point
Race car engine	1.6	Feature	
Sedan engine	2.2	Feature	
EngineSummary	B_15972_EngineSummaryOptic	Family	String
DemoCar_Entertainement	B_15972_DemoCar_Entertaine	Family Group	
DemoCar_Exterior	B_15972_DemoCar_Exterior	Family Group	
DemoCar_FilterOptions	B_15972_DemoCar_FilterOptic	Family Group	
Interior	B_15972_DemoCar_Interior	Family Group	
SafetyOption	B_15972_DemoCar_SafetyOpt	Family Group	
Transmission	B_15972_DemoCar_Transmissi	Family Group	
Others	Other configurations	Family Group	
Warranty	Car Warranty	Family	Integer
2 years	2	Feature	
3 years	3	Feature	
5 years	5	Feature	
Key Milestone dates	Dates	Family	Date
Product Release Date	01-Sep-2019	Feature	
Crash Test Date	30-Sep-2019	Feature	

For example, you created features for different colors, such as **Red**, **Green**, and **Blue**. As the product develops, you need to change them to **Cherry Red**, **Forest Green**, and **Aqua Blue**, respectively. You can edit their names and work with them as primary attributes. IDs remain the same and you cannot edit them.



For numeric and date families, you do not have to remember the actual numeric values or milestone dates in your organization. You can search and create expressions using the corresponding user-friendly business names, such as **Product Release date** instead of actual dates.

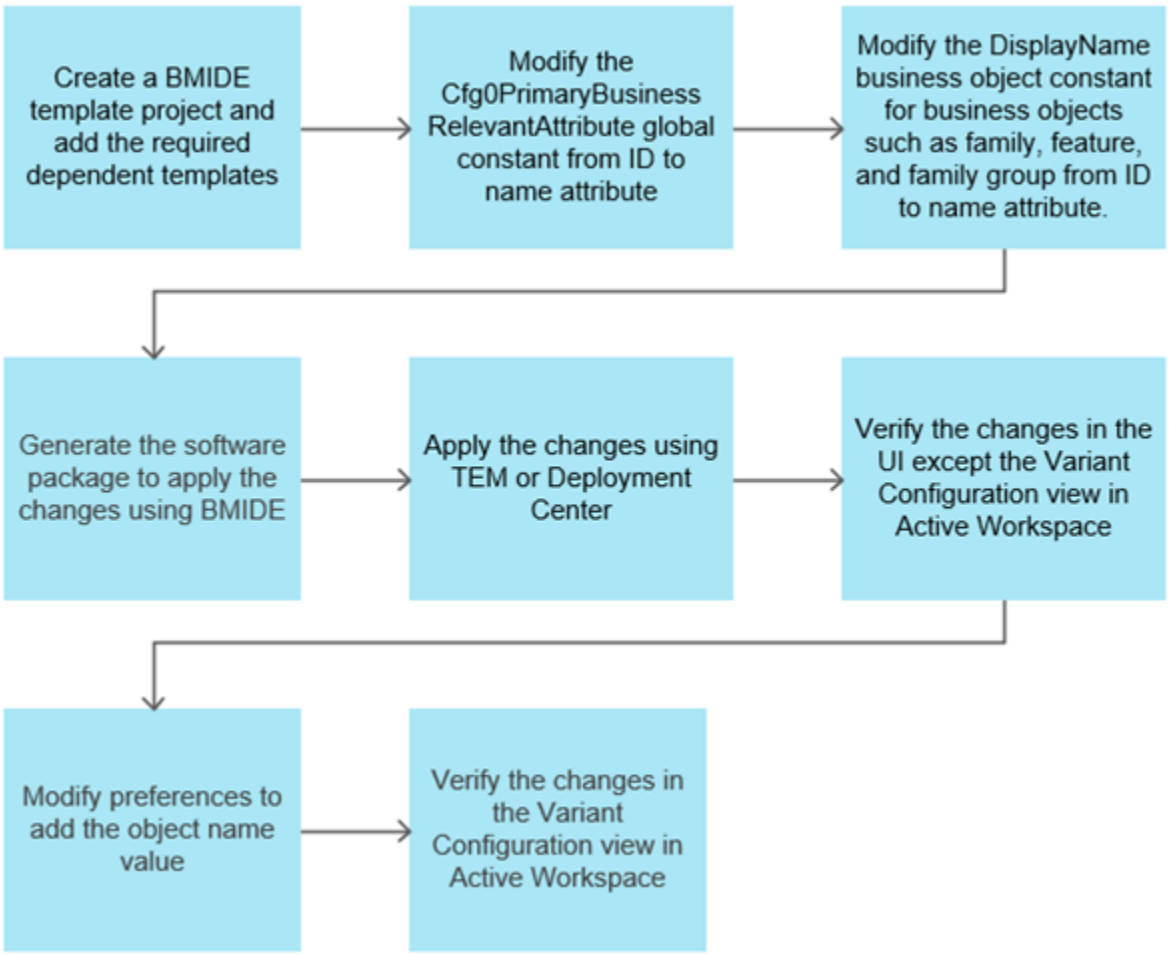
If you are using a numeric feature, such as an integer or a float, the feature ID and the feature name must be the same. For example, if a user provides only a feature ID, then you must also populate the feature name with the same ID value and vice versa. If a numeric feature ID does not match a feature name, then the system generates an error and the create operation fails.

Set name or ID as the primary business attribute

The name is used as the primary business attribute by default. It appears as the default in all Product Configurator dialog boxes and views such as **Variability Explorer**, **Configurator Rules**, **Variant Expression Editor**, **Availability Matrix**, **Saved Variant Rules**, and **Variant Configuration**.

To change the default to the *ID* attribute in the dialog boxes and views, you must modify the **Cfg0PrimaryBusinessRelevantAttribute** global constant and also the **DisplayName** attribute for business objects such as family, feature, and group in BMIDE.

In addition to the BMIDE changes, you must modify site preferences to display the name as the primary business attribute in the **Variant Configuration** view in Active Workspace.



1. Create the BMIDE template project
2. Modify the Cfg0PrimaryBusinessRelevantAttribute global constant
3. Modify the DisplayName business object constant for family, feature, and group business objects
4. Generate the software package to apply the changes using BMIDE
5. Apply the changes using TEM or Deployment Center
6. Verify the changes in the user interface (UI) except the Variant Configuration view in Active Workspace
7. Modify preferences to add the object name value
8. Verify the changes in the Variant Configuration view in Active Workspace

Display the name attribute in dialog boxes and views

> Note:
>
> The following procedure shows how to set the name as the primary business attribute. Similarly, you can set ID as the primary business attribute.

1. Create a BMIDE template project.

   a. Launch Business Modeler IDE (BMIDE).

   b. Create a new BMIDE template project, for example, **a5nameid**.

   c. In **Dependent Templates**, search for **cfg** and select **Teamcenter Configurator**.

   d. Search and select all other dependent templates and click **Finish**.

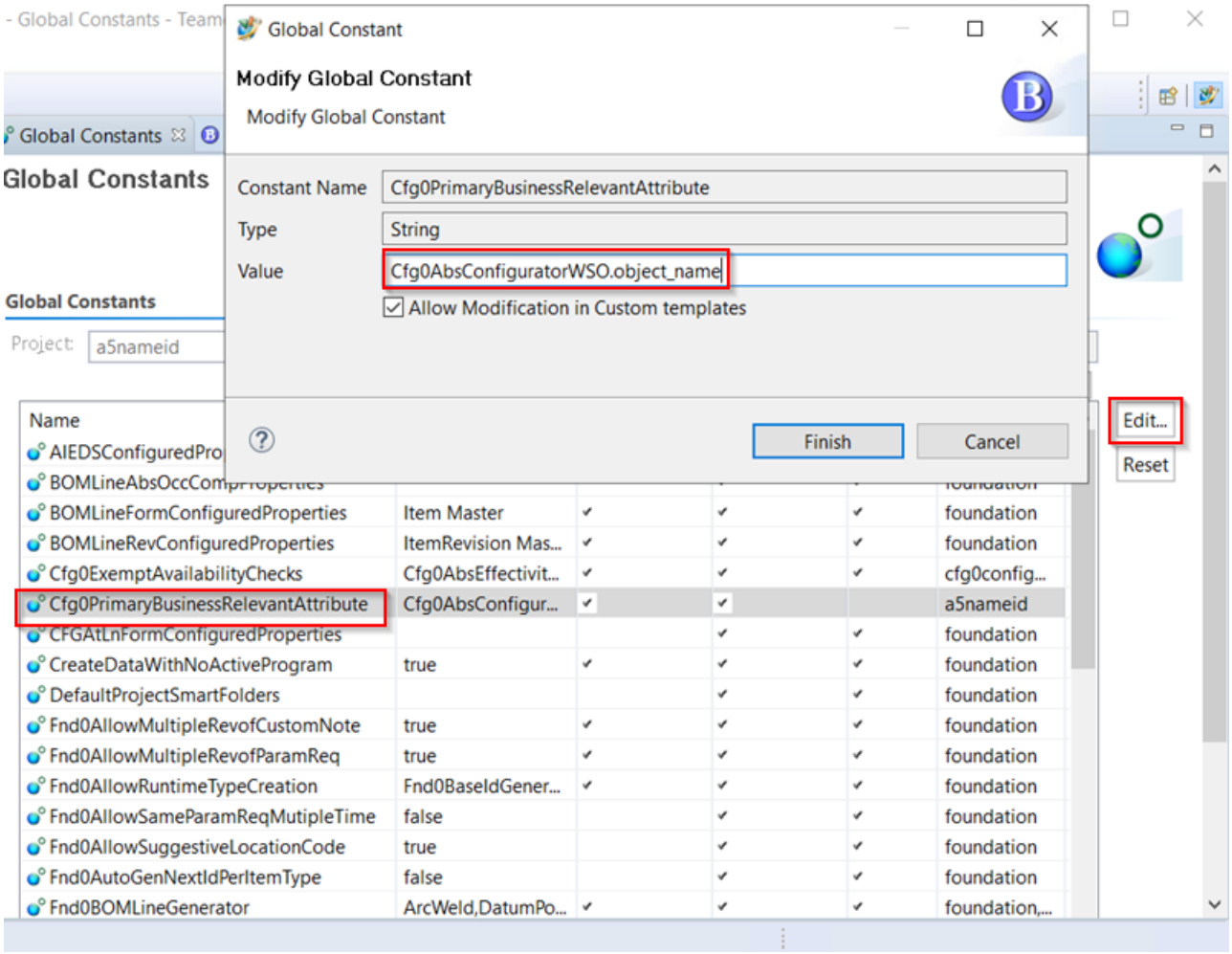
2. Modify the **Cfg0PrimaryBusinessRelevantAttribute** global constant.

   a. In **Business Objects**, select the template project you created, and click the **Open Global Constants Editor** icon.

   b. Select the **Cfg0PrimaryBusinessRelevantAttribute** global constant and click **Edit**.

      By default, the value is **Cfg0AbsConfiguratorWSO.cfg0Object_name**. It means that the name is used as the primary business attribute.

      The valid value for ID is **Cfg0AbsConfiguratorWSO.cfg0ObjectId** and that for name is **Cfg0AbsConfiguratorWSO.object_name**.



c. To change the default business attribute to name, change the value to **Cfg0AbsConfiguratorWSO.object_name** and click **Finish**.

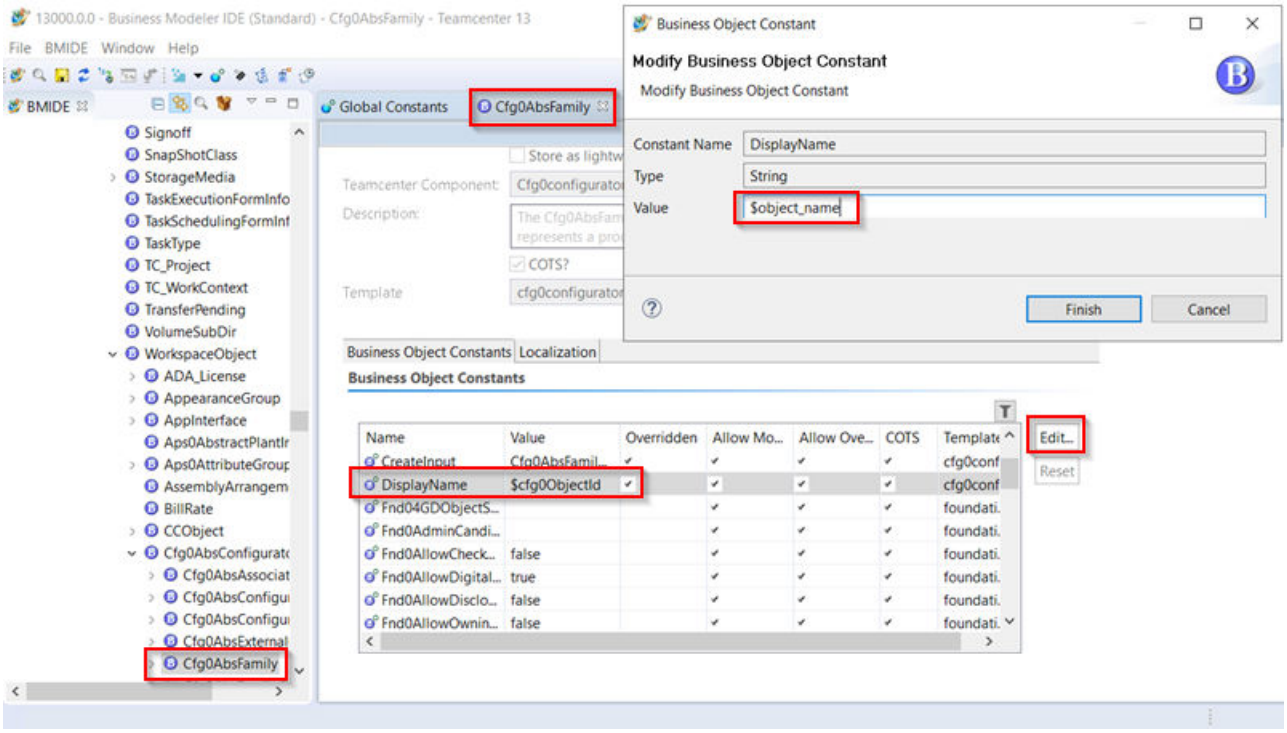
3. Modify the **DisplayName** attribute for business objects such as family (**Cfg0AbsFamily**), feature (**Cfg0AbsValue**), and group (**Cfg0AbsFamilyGroup**).

   The value of **Cfg0PrimaryBusinessRelevantAttribute** must be in sync with the value of the **DisplayName** business object constant of Product Configurator business objects. If you update the value of the **Cfg0PrimaryBusinessRelevantAttribute** global constant to **Cfg0AbsConfiguratorWSO.object_name**, then you must update the **DisplayName** business object constant of configurator business objects also to **object_name**.

   a. In BMIDE, click **Find Element**, type **Cfg0AbsFamily** to search for it, select it, and click **Open in Editor**.

   b. In the **Business Object Constants** tab, select the **DisplayName** attribute, and click **Edit**.

   c. Change the default value from **$cfg0ObjectId** to **$object_name**.



d. Repeat **step a** to **step c** for other business objects such as feature (**Cfg0AbsValue**) and group (**Cfg0AbsFamilyGroup**).

4. Generate the software package to apply the changes.

   a. Click **BMIDE→Save Data Model**.

   b. Click **BMIDE** > **Generate Software Package** and click **Next** in the **Recommendations for Backing up Your Template Project Source Files** dialog box.

   c. In the **Generate Software Package** dialog box, note down the path in **Target folder** and click **Finish**.

      Example:

      *BMIDE***workspace\13000.0.0***a5nameid**\output\wntx64\packaging\full_update***a5nameid_**wntx64_1.0_13.3**

5. Apply the changes using TEM.

   a. Run TEM as an administrator.

   b. Click **Next** until you reach the **Features** panel.

c. Click **Browse** and navigate to the location of the **Target folder** that you had noted down earlier.

d. Select the custom template under **Extensions**, for example, **a5nameid**.

e. Stop the pool manager service.

f. Click **Next** until you reach the **Confirmation** panel and then click **Start**.

g. Restart the pool manager service when the TEM feature installation process is complete.

6. Verify the changes in Product Configurator in the rich client.

a. In Product Configurator, search for a configuration object and double-click it to open the **Variability Explorer** view.

b. Verify that the **Name** column is the first column in the **Variability Explorer** view. Prior to modifying the global constant, the **ID** column was displayed.

c. Select a BOM line and modify the **Name** attribute, for example, change **PL01** to **PL01_name**.

d. Click **Open Variant Configuration view**.

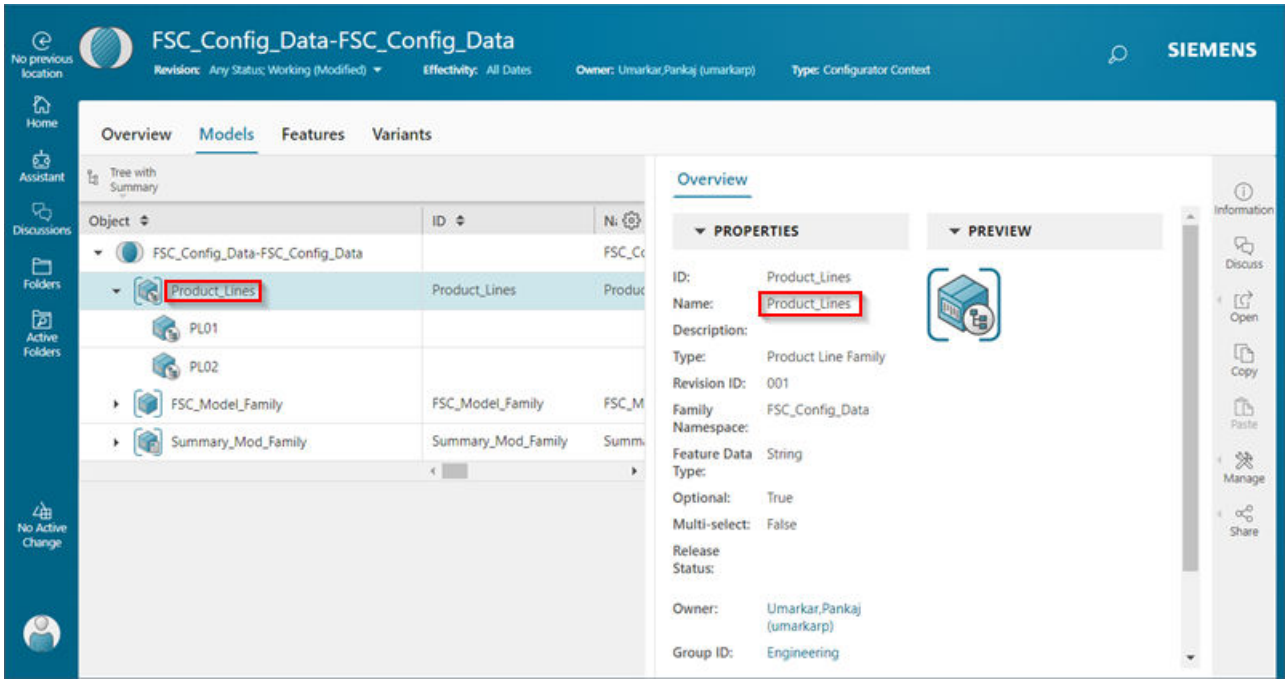
The BOM line you modified is displayed with the **PL01_name** in the **Variant Configuration** view.

If you have not changed the primary business attribute as name by making the BMIDE changes, the name remains unchanged as **PL01** in the **Variant Configuration** view even if you modify the **Name** attribute.

7. Verify the changes in Active Workspace.

a. In Active Workspace, open a configurator context, and click the **Models** tab.

b. Select **Tree with Summary**. In the **Overview** section, verify that the **Name** value is the same as the **Object** value.



Display the name attribute in the Variant Configuration view in Active Workspace

To change to the name attribute in the **Variant Configuration** view in Active Workspace, you must modify two site preferences by adding the **object_name** value to them in addition to **changing to the name attribute using BMIDE**.

1. Perform the procedures described in **edit the Cfg0PrimaryBusinessRelevantAttribute global constant value** and **edit the DisplayName attribute for business objects** in BMIDE.

2. In Teamcenter, edit the **Cfg0AbsConfiguratorWSO.CellProperties** and **Cfg0AbsValue.CellProperties** site preferences to add the **object_name** value or replace **Cfg0ObjectId** with **object_name**.

   For cell rendering in Active Workspace, each object type is controlled by a preference. If no preference is found for a specific type, then the closest parent that contains the preference value is considered. For example, if **Cfg0AbsValue** has a value for this preference value, it is considered. Otherwise, the parent, **Cfg0AbsConfiguratorWSO** that has the preference value is considered.

   For more information about defining properties that are displayed in object cells, see *Teamcenter Administration*.

3. Verify the changes in Active Workspace.

   a. In Product Configurator, search for a configuration context and double-click it to open the **Variability Explorer** view.

b. Select a BOM line and modify the **Name** attribute, for example, change **PL01** to **PL01_name** and save it.

c. In Active Workspace, open a structure to which the configuration context is attached and click the **Variant Configuration** view.

d. Verify whether the **Name** attribute is changed, for example, from **PL01** to **PL01_name**.

Utilities and preferences to administer effectivity data

In addition to the general Product Configurator preferences, you can use the following utilities and preferences to administer effectivity data:

Utilities

- **manage_effectivity_options** utility

  Allows you to create and manage effectivity options for the product or program.

- **validate_revrule_effectivity** utility

  Allows you to validate the effectivity criteria on specified revision rules, generate validation records, and associate them with the revision rules.

- **upgrade_variant_rule_data** utility

  Migrates the variant rule objects to the new expression data model that allows to persist the session information.

- **save_variant_rule_as_variant_criteria** utility

  Saves existing variant rules as new variant criteria objects.

For more information about using these utilities, type *utility_name*-**h** on the Teamcenter command prompt.

Preferences

- **TC_configurator_relationship** preference

  Allows you to define the relationship Teamcenter navigates from the product design to a related item revision. It does this to derive information about the product name (defined in the **fnd0VariantNamespace** item property) and product namespace (item revision ID) from the product item revision. This preference is not used if you work with an external product configurator.

- **PCA_effectivity_shown_columns**

Specifies if unit, date, or all columns are shown in the **Effectivity** view of Product Configurator.

- **TC_EffectivityConfigurable_MaxCacheSize**

  The Teamcenter server caches effectivity data for configurable objects to prevent repeated frequent database queries for the same set of configurable objects. Use this preference to limit the size of the cache to optimize the balance between memory footprint of the **tcserver** process and the performance impact of an increased database SQL query count.

  The cache is automatically purged on a last-accessed basis. Cached values are automatically refreshed for loaded and up-to-date objects when a cached value is accessed and the cached value has changed since it was cached.

- **TC_Fnd0Booleansolve_EffectivityDateRangeFromTo**

  You can display effectivity date ranges in one of two ways:

  - Inclusive date ranges, where the end date is included (a from/to paradigm). Objects whose effective date range displays in a format such as **Date=2011-04-01..2011-04-30** *are* effective until the end of the last day, in this case, **2011-04-30**. Similarly, if the date range displays as **2011-04-01..2011-04-30T17**, the objects are effective until the end of hour 17 on the specified day, **Date=2011-04-30**. Set this preference to **TRUE** to use this display format.

  - Half-open ranges, where the end date is excluded (an effect in/effect out paradigm). Objects whose effective date range displays in a format such as **Date=2011-04-01..2011-05-01** are *not* effective for the end date, in this case, **Date=2011-05-01**. Set this preference to **FALSE** to use this display format.

  Dates must be formatted to ISO 8601 using the server time zone. For example, a display value of **2011-04-30T17** means **2011-04-30T15:00:00Z** if the Teamcenter server is located in the MET time zone.

- **TC_Fnd0Booleansolve_EffectivityIntegerRangeFromTo**

  You can display effectivity date ranges in one of two ways:

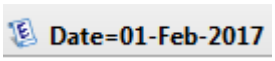
  - Inclusive unit ranges, where the end value is included (a from/to paradigm). Objects whose effective unit range displays in a format such as **Unit=1..10** *are* effective for the end unit, in this case, **10**. Set this preference to **TRUE** to use this display format.

  - Half-open ranges, where the end value is excluded (an effect in/effect out paradigm). Objects whose effective unit range displays in a format such as **Unit=1..10** are *not* effective for the end unit. Set this preference to **FALSE** to use this display format.

- Users can set *point effectivity* on the effectivity hyperlinks on the header of the **Variability Explorer** view and **Variant Configuration** view irrespective of the values set on the **TC_Fnd0Booleansolve_EffectivityDateRangeFromTo** preference. However, if you are setting

effectivity using the **Effectivity** view, then the values set on the above mentioned Teamcenter preferences are honored.

The following shows an example of how the date effectivity is displayed based on **TC_Fnd0Booleansolve_EffectivityDateRangeFromTo** values.

Views	= true	= false
Variability Explorer or **Variant Configuration** header		
Effectivity view		*Cannot* set point effectivity.

- **TC_Fnd0SoaConfigFilterCriteria_AutoPublishRevisable**

  Effectivity and variant data for a persistent object manager (POM) revisable object can change only if the object is in a private edit state. The **Fnd0SoaConfigFilterCriteria** SOA service that sets an effectivity or variant condition automatically puts revisable objects into a private edit state. If this preference is set to **TRUE**, revisable objects that are put into private edit state by a **Fnd0SoaConfigFilterCriteria** service call are automatically published by the same service call. Consequently, the revision number increments for each **Fnd0SoaConfigFilterCriteria** service call. Effectivity and variant data changes are immediately visible to other users if they have read access. Revisable objects that are already in a private edit state before the system invokes the **Fnd0SoaConfigFilterCriteria** service call remain in a private edit state and are not automatically published.

  Depending on your business process, consider setting this preference to **FALSE**, allowing users to accumulate multiple changes for the revisable object. They can then publish all the changes in a single action to other users with read access.

Configure the display of variant expressions

The displayed format of variant expressions is configured with the following site preferences:

- **TC_show_family_namespace_prefix**

  Enables the display of the family namespace within the expression string.

- **TC_show_option_family_prefix**

  Enables the display of the families within the expression string.

Use these preferences to configure the following display formats:

TC_show_family_ namespace_prefix	TC_show_option_ family_prefix	Expression format
true	**true**	[*namespace*]*family* = *value*
false	**true**	*family* = *value*
true	**false**	[*namespace*]*value*
false	**false**	*value*

Character representations:

- & is represented by AND

```
[NS]A = a1 AND [NS]B = b1
```

- | is represented by OR

```
[NS]A = a1 OR [NS]A = a2
```

- NOT is represented by !

```
[NS]A != a1
```

- Date values are displayed in the standard Teamcenter locale-specific date format.
- Range expressions are displayed using inequality operators

```
[NS]A = a1 AND [NS]LENGTH <= 50
[NS]A = a1 AND [NS]LENGTH <= 50
[NS]A = a1 AND [NS]LENGTH >= 20
```

- Multiple column expressions are grouped in parentheses

```
([NS]A = a1 AND [NS]B = b1) OR ([NS]A = a2 AND [NS]C = c1)
```

- Multiple selections within a family are grouped in parentheses

```
([NS]A = a1 OR [NS]A = a2) AND [NS]B = b1
```

- Optional families support **NONE** or **ANY**

```
[NS]A = a1 AND [NS]D = NONE (same as [NS]D = ‘’)
[NS]A = a1 AND [NS]D = ANY (same as [NS]D != ‘’)
```

Control access to configuration data

You can use the appropriate privileges to control user access for adding or removing data from a configurator item. Teamcenter checks that the user has the required privileges on the configurator item when any of the following actions are performed.

User action	Required Access Manager privileges
Allocate a group, family, or feature to a configurator item	**Add Content**
Deallocate a group, family, or feature from a configurator item	**Remove Content**
Create a configurator rule	**Add Content**
Delete a configurator rule	**Remove Content**
Relate a configurator rule to a configurator item	**Add Content**
Remove relation between a configurator rule to a configurator item	**Remove Content**

Share configurator data

You can share configurator data by using:

- Multi-Site Collaboration
- **Briefcase** data transfer with the high level TC XML format to transfer objects offline
- TC XML low level commands.

Configurator context and associated objects, such as groups, families, features, model families, product models, summaries and their members, packages and their members, constraints, defaults, and saved variant rules can be synchronized to the target site. Ownership of the summaries and packages must be transferred to the target site when the ownership transfer operation is performed.

Global rules are not exported with configurator contexts. For deployments with global rules, export/import of a dictionary is mandatory.

You can use Multi-Site Collaboration and TC XML to share configurator data as follows:

- Replicate unconfigured configurator data between sites.
- Synchronize configurator data between sites.

- Transfer ownership of configurator data between sites.

To enable the **Briefcase** data transfer, your administrator must set the following settings in both the source and the target sites:

- In the **Organization** application, ensure the **Uses TC XML Payload** and **Is Offline** check boxes are selected for the sites.

- Set the **GMS_offline_use_TcGS** site preference to false.

Summary of data sharing support in Product Configurator

Data shares scheme	Supported	Not Supported

Multi-Site Collaboration	Command line utilities:
	To replicate data, use the **data_share** utility
	Example:
	`data_share -u=<userid> -p=<password> -f=send -item_id= <ConfiguratorContext/ Dictionary ID> -site=<target_site_id> -optionset= MultiSiteExpOptSet`
	To synchronize the replicated data, use the **data_sync** utility
	Example:
	`data_sync -u=<userid> -p=<password> -sync -update -item_id= <ConfiguratorContext/ Dictionary ID> -site=<target_site_id> -force -optionset=MultiSiteExpOptSet`
		Rich client operations, such as:
		• Remote check-out
		• Publish
		• Unpublish
		• Register
		• Unregister
		• On-demand-sync
		Command line utilities:
		• **item_import**
		• **item_export**
		• **distributed_execute**
		• **export_recovery**
		• **sync_on_demand**

High level TC XML format	Rich client operations, such as: • To replicate and synchronize the data, use the **Briefcase** data transfer.	Not applicable

Data shares scheme	Supported	Not Supported

	• To replicate, synchronize, and transfer ownership of data, use the following command line utilities:
	
	• **tcxml_export**
	
	Example:
	
	`tcxml_export -u=<userid> -p=<password> -file=<file_name>.xml -optionset=TIEUnconfiguredExportDefault -uid=<uid_of_ConfiguratorContext/Dictionary> -targetsites=<target_site_id>`
	
	• **tcxml_import**
	
	Example:
	
	`tcxml_import -u=<userid> -p=<password> -file=<file_name>.xml`	

Low level TC XML	To replicate, synchronize, and transfer ownership of data, use the following command line utilities:
	
	• **tcxml_export**
	
	Example:
	
	`tcxml_export -u=<userid> -p=<password> -file=<file_name>.xml -optionset=TIEUnconfiguredExportDefault -uid=<uid_of_ConfiguratorContext/Dictionary> -targetsites=<target_site_id> -low_level -force_retraverse`
	
	• **tcxml_import**
	
	Example:
	
	`tcxml_import -u=<userid> -p=<password> -file=<file_name>.xml -low_level`
		Not applicable

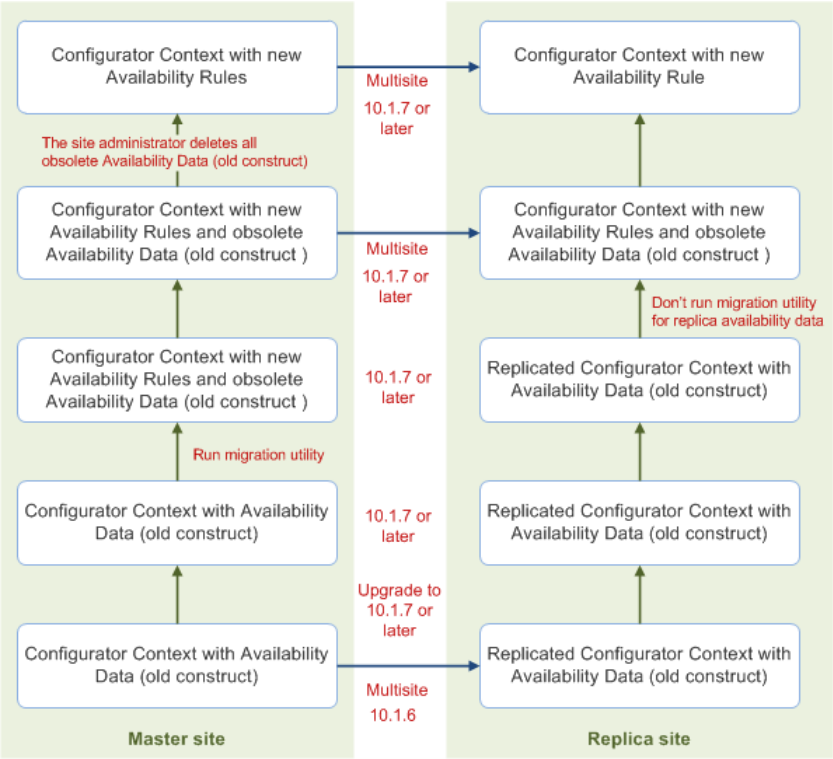
PLM XML	Not supported	Not applicable

Note:

In this version of Teamcenter, the use of the TC XML low level features of the **data_share** and **data_sync** command utilities is restricted and requires the **SITCONS_AUTH_KEY** environment variable set to a valid key value.

TC XML data exchange compatibility

The Teamcenter configurator TC XML data cannot be exchanged across all Teamcenter versions. This is a general limitation of TC XML data exchange whenever there are major data model changes across Teamcenter versions. Because Product Configurator on Rich Client — Usage in Teamcenter 10.1.7 has major data model changes, any TC XML data exported before Teamcenter 10.1.7 cannot be imported to Teamcenter 10.1.7 or later.



You must select one of the following as a root object of configurator data sharing to ensure that all associated objects are exported:

- Configurator context
- Configurator dictionary

Configurator does not support data exchange of other configurator objects such as a family, feature, product model, or rule as a root object.

You can create and allocate additional features, feature families, and groups to a configurator context based on the **ADD_CONTENT** and **REMOVE_CONTENT** privileges. The same privileges are also required when adding or removing rules from a configurator context.

If you need to change features, feature families, and groups, you must have the **WRITE** privilege for these objects. Thus, these objects can only be updated at the owning site.

- The **WRITE** privilege is not required for the configurator context.

  You can therefore create and allocate additional objects to the replicated configurator context if the **ADD_CONTENT** and **REMOVE_CONTENT** privileges are granted to them.

If your business process requires authoring data at multiple sites, or your site may need to exchange configurator data with another Teamcenter site in the future, Siemens Industry Software Inc. recommends to attach a naming rule to the ID property of association threads and threads for generic configurator rules, such as **Cfg0AbsAssociationThread.fnd0ThreadId** and **Cfg0AbsRuleThread.fnd0ThreadId**.

Note:

Siemens Industry Software Inc. recommends that you maintain a single master site for authoring and modification of your configurator data. You can then replicate your data to other sites.

Warning:

If your site contains Product Configurator data with a large number of configurator rules, such as constraint rules or variant rules, it also contains a large number of expression objects. TC XML high level import of a large set of data can encounter scalability and performance issues.

Performing calculations

About external calculations

Your business process may require you to calculate parameters during configuration. The configuration process may require string processing operations, mathematical functions, or complex algorithms that cannot be expressed as native rules within Product Configurator.

Your company may already have existing programs for custom calculations that need to be leveraged during the configuration process. In the current release of Teamcenter, you use the *external calculations* functionality to include such operations during the configuration process.

For example, the following scenario uses a fictitious company that delivers highly-configurable industrial cranes to customers all over the world. This company has developed proprietary and legacy knowledge systems to calculate the optimal number of pulleys and the preferred wire rope diameter and length based on various requirements, such as lift capacity, working height, and support points. Because these calculations are complex and non-linear, you cannot precalculate these values and save them in the

system as logical Product Configurator rules. Instead, Teamcenter calls those external systems to perform calculations during the configuration solve process.

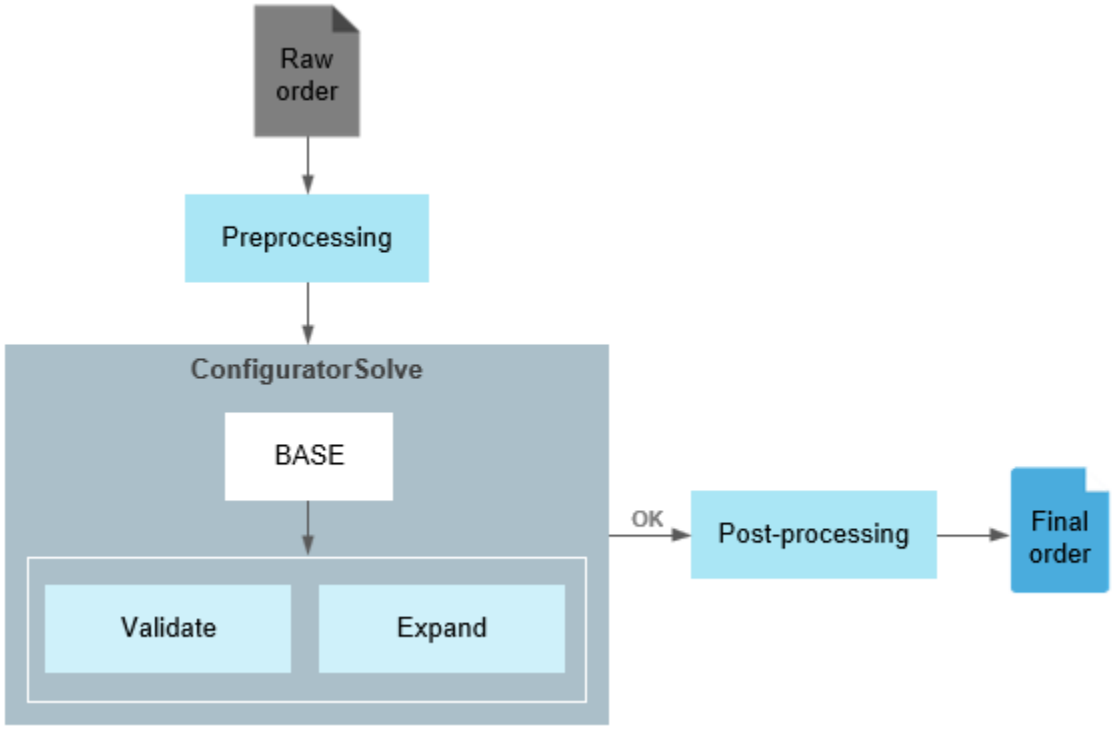
External calculations can be performed before or after the configurator solve. Calculations performed before the solve are used to calculate values for feature families. Then, those feature families participate in the solve. During the solve, the system applies configurator rules that refer to the participating feature families. Calculations performed after the solve are used to calculate results that consume values from feature families configured during the solve process.

The process of invoking external operations from Product Configurator involves:

- Marshalling the feature families and values that are needed by the external calculation into XML
- Sending that XML to a Web service that performs the calculation
- Unmarshalling the returned XML and setting feature values based on the solve results

> Note:
>
> External calculations are only available in the four-tier installations of Teamcenter. They are not supported in the two-tier clients.

Calculation phases



The external calculation and processing of functions during the Product Configurator solve process consist of multiple phases:

- **Preprocessing (PRE)**

  During this phase, you may leverage external or legacy calculation functions before the solve is initiated by using any feature families set by the user or imported as an input and making the calculation results available to the solve.

  > Note:
  >
  > If a preprocessing calculation results in an error, the order validation and expansion processes do not occur in the system.

- BASE

  During this phase, the order string validation and expansion solve operations are performed in the system.

- **Post-processing (POST)**

  During this phase, after the successful completion of the solve, you may leverage external or legacy calculation functions using an input containing any feature family values determined during PRE calculations or during the solve. The calculation results supplement the feature family values from the solve.

  > Note:
  >
  > If the order is determined to be invalid during the solve, the post-processing calculations do not take place.

PRE and POST processing calculations are not directly executed in Teamcenter. They are only managed in Teamcenter. A separate, customer-provided calculation service performs the actual calculation. Teamcenter sends the inputs to the calculation service. When the results are returned to Teamcenter, it updates the corresponding feature families.

The following is required for the customer-provided calculation service:

1. A custom calculation service (Web service with RESTful API) that directly includes the calculations, algorithm, and optional logic. It subsequently may call one or more services that perform the calculation. It can also be a wrapper that calls a calculation program, a library, or a function

2. A capability to unmarshal the XML to obtain feature values required as inputs and determine the features that must be calculated and returned as outputs

3. If required, a capability to verify the calculation request is authentic based on a JWT token sent from Teamcenter

4. A capability to marshal an XML that includes the calculation outputs and, optionally, a list of error conditions and explanatory messages.

Within the PRE or POST phases, multiple external calculations may be executed in a specified sequence. The system evaluates any failure returned from the external operation based on the specified severity.

> Note:
>
> In the current release, the system considers all failures as errors, and once encountered, it stops the operation.

XML interaction with external calculation operations provide and accept features as follows:

- Features introduced during the PRE phase are handed over to the solve to be validated. They are presented to the user when viewing the variant criteria.
- Features introduced during the POST phase are presented to the user, but they are not revalidated by the system.

> Note:
>
> The calculation service is called during the PRE or POST phases of the Product Configurator solve operation. This calculation requires additional processing. Depending on the complexity of the external calculation, it may impact the solve performance.
>
> The security of the data is controlled by the custom code. The code must adhere to the Teamcenter access control mechanism and not violate security of any information.

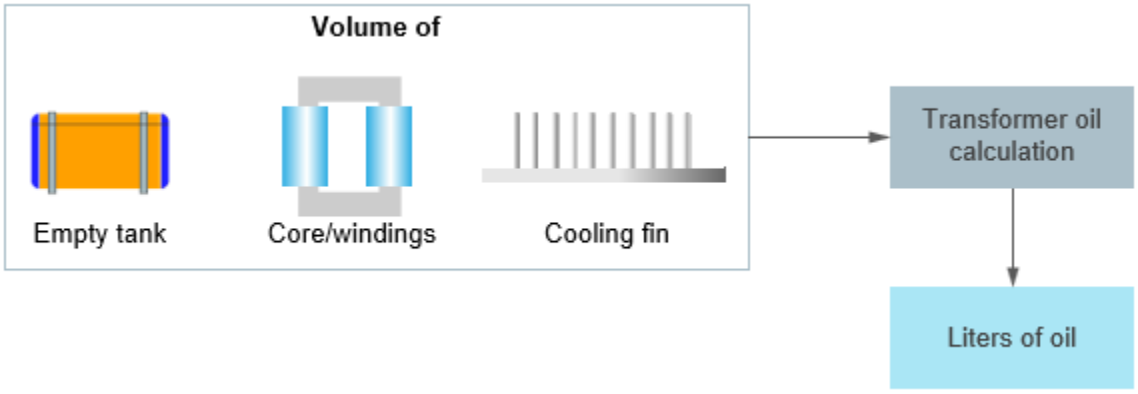
The initiation (PRE) phase example

The following represents the conversion of an imported raw order string, sometimes referred to as a line string, into individual feature families suitable for consumption in Product Configurator.

In this example, the external calculation uses the feature family that contains the raw order string as an input. It splits the delimited string apart and returns the values for individual feature families as outputs. Then, the system proceeds with the standard validation and expansion process.

The POST processing example

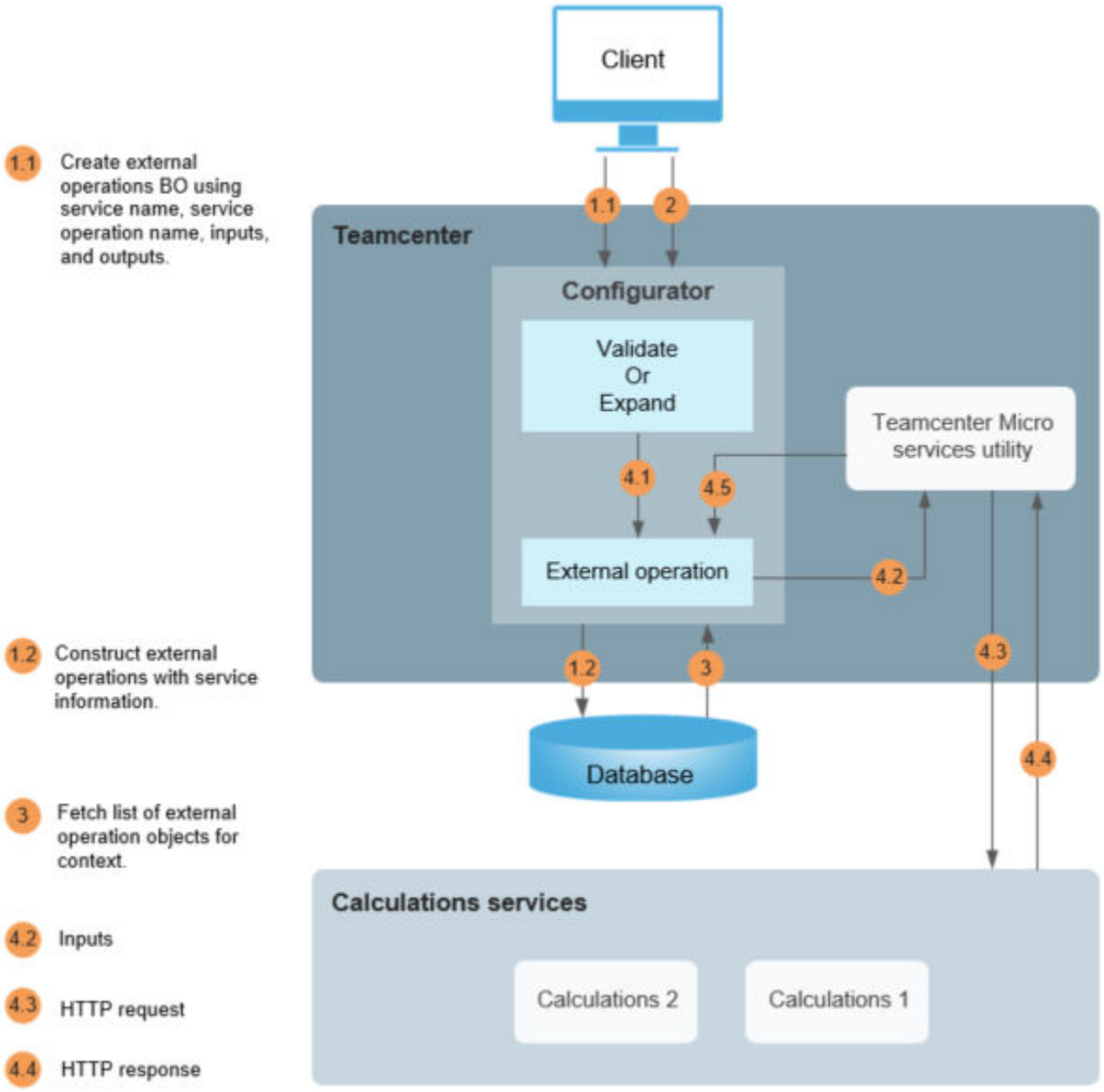
The following displays the quantity (non-occurrence) calculation.



In this example, the external calculation consumes the volumes determined during the solve as an input and provides the value for the liters of oil feature family.

External calculations process flow

An external operation is facilitated by calling customer calculation service RESTful APIs using the microservice framework with published RESTful APIs.



- (1) During initial setup, an administrator creates one or more new calculations services with published RESTful APIs.
- (2) A configurator administrator creates one or more external operation objects using standard metaframework APIs and specifies the following:
  - The service end point and the service operation name created in the initial step
  - The list of input and output feature families

  A configurator administrator creates one or more **External Operation Block** objects for PRE and POST phases, associates the required operation objects to it, and specifies the required order of execution within the block.

- (3) & (4) A configurator user clicks **Validate** or **Expand** in the **Variant Configuration** view.

  The system assesses if the **External Operation Block** and operations are registered to the current configurator context. If so, it retrieves the required information about the operations.

- (5) The system uses the service end point and the service operation name of registered operations to invoke calculation service. The inputs are configured by users and outputs are consumed by the Product Configurator solve operation, depending on the phase of the operation.

Setting up external calculations in Product Configurator

Prior to calculating parameters during configuration in Product Configurator, you need to develop your own calculation Web service which may be called to perform external operations.

Create custom **HTTP-based RESTful APIs**, such as:

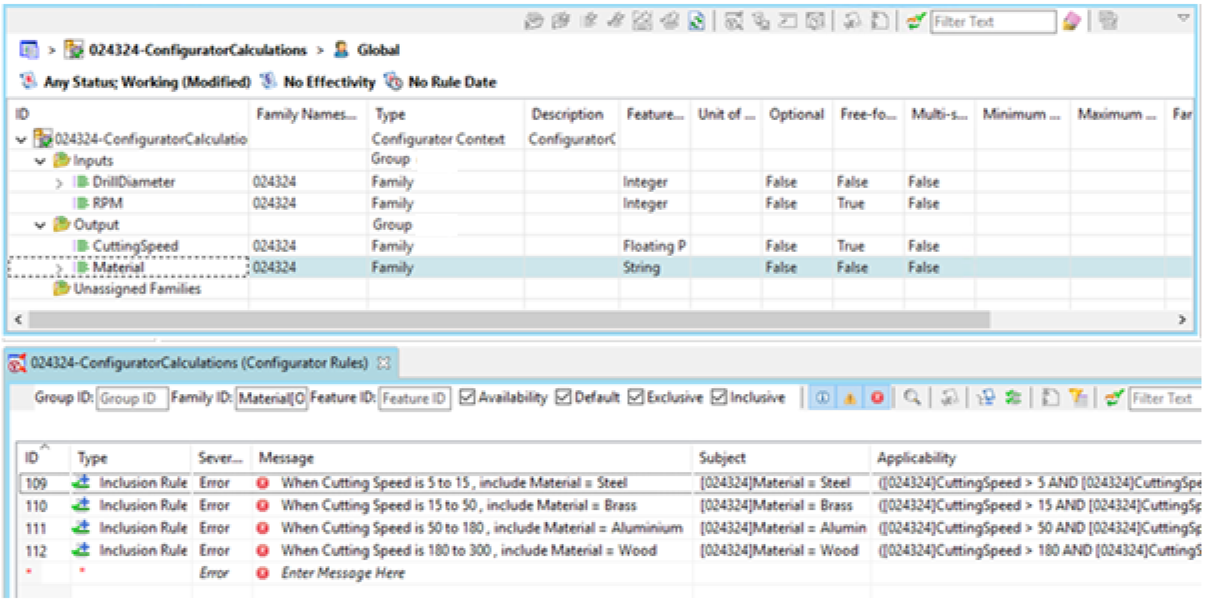
- A base URI, for example, `http://xyz.com/calculations/`

- Standard HTTP methods, for example, GET, POST, PUT, PATCH and DELETE

The RESTful API needs to accept the content type in XML. The request body contains the request XML. Product Configurator invokes the RESTful API using the HTTP request.

For example:

In the following scenario, Teamcenter uses **Drill Diameter** and **RPM** (rotational speed) as inputs to calculate **Cutting Speed** using an external calculation. The formula to determine the cutting speed is `(Diameter*RPM*Pi)/12`. Then, the system uses configurator rules to determine **Material**. The **Material** constraints require **Cutting Speed**, therefore, the calculation proceeds with the PRE calculation phase.



Suppose your company Web service URL is `127.0.0.1:5000` and the RESTful API is `/configuratorcalculations/v1/cuttingspeed`.

```
POST /configuratorcalculations/v1/cuttingspeed/
cc6e6ce40e4aecb92d193086543fbc3b7e7748f4c3982548461d13af4ed6acc3 HTTP/1.1
Host: 127.0.0.1:5000
Accept: */*
Accept-Encoding: deflate, gzip
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpc3MiOiJ0Y3NlcnZlciIsInN1YiI6IkNvbmZpZ3VyYXRvckNhbGN1bGF0aW9ucyJ9.
R9KoC6Gg8kiCUwLLakJALIeVzT7MoyNilUaU8yMBfay0wa0PYEpFMT9xlUUdVM_
zuy4cwjaqYwZ9rX898P1sv1TmjRpKpo38BhSdNxI3KFhtZ0nGWXszDusfu4VUElH2eR2AJSh2QXtUp7uES9Bas6THD
lUzcd9YMvQn4q9JLIHH1C_UM7mKS99dQjeOzhOsk9iOhVNKi3DnwAe14oWxin-hMX5QcJriYBp-CEaCh3IQBW6Aui
JMOd5vmwfzvbpwkGxeZE2mnjAi5DDQ17gT1__8INkmiY2D99zaCPtkPrTIKbwulTK4wdm-
                                    CsiMTcCU5G7iSSzeSJWp5188cvN0Ew==
Content-Type: application/xml
Content-Length: 786
```

This HTTP Request comprises the following:

- Request body
- Authentication

Request body

It contains a request in the XML format with the **ExecuteOperationRequestType** schema. The response from the calculations Web service is using the **ExecuteOperationResponseType** schema.

```
<!--
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
```

```
  <!-- External Operations Request:
       This schema is used to send the external operation's input
parameters as
       XML while invoking the external operations.The input parameters such
as
       input families,its values and requested families are expected to be
       populated as per this schema while invoking the external operation.
       As inputs to this request, it takes input families as featureFamily,
       values as featureValue and expected families as featureFamily. The
       operation name can be specified for information that which external
       operation is being invoked. In the external operation, these values
may
       be used to calculate outputs. The external operation is expected to
       return calculated values for requested families
-->
  <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-->
  <xsd:complexType name="ExecuteOperationRequestType">
    <xsd:complexContent>
      <xsd:attribute name="operationName" type="xsd:string" use="optional">
        <xsd:annotation>
          <xsd:documentation>
            Name of the external calculation operation to be executed by
custom
            code.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:attribute>
      <xsd:element name="featureFamily" type="FeatureFamilyType"
minOccurs="0"
            maxOccurs="unbounded">
        <xsd:annotation>
          <xsd:documentation>
            The list of feature families for which value permutations are
            available.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name="featureValue" type="FeatureValueType"
minOccurs="0"
            maxOccurs="unbounded">
        <xsd:annotation>
          <xsd:documentation>
            An optional list of feature values. If the request includes one
            or more feature values of a featureFamily, the remaining values
in
            this family are ignored when generating the value permutations.
          </xsd:documentation>
        </xsd:annotation>
```

```
      </xsd:element>
      <xsd:element name="requestedFeatureFamily" type="FeatureFamilyType"
                minOccurs="0" maxOccurs="unbounded">
        <xsd:annotation>
          <xsd:documentation>
            The list of feature families for which value permutations are
            requested.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-->
  <!-- External Operations  Response
-->
  <!-- This schema is used to receive the external operation's output
parameters
  as xml while invoking the external operations. The output parameters such
as
  requested families and its values are expected to be populated as per
this
  schema while sending calculated values to the caller of external
operation.
  This response takes expected families and values as featureFamily and
  featureValue as inputs to this response. The response includes the status
  (StatusType ) of external function executed. This is used to populate
success
  or failure of external function along with related error messages.
  The operation name can be specified for information that which external
  operation is being invoked.   -->
  <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-->
  <xsd:complexType name="ExecuteOperationResponseType">
    <xsd:complexContent>
      <xsd:attribute name="operationName" type="xsd:string" use="optional">
        <xsd:annotation>
          <xsd:documentation>
            Name of the external calculation operation executed by custom
code.
            This is for the information to know response is coming from
which
            external operation.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:attribute>
      <xsd:element name="featureFamily" type="FeatureFamilyType"
minOccurs="0"
```

```
                    maxOccurs="unbounded">
        <xsd:annotation>
          <xsd:documentation>
            The list of feature families for which value permutations are
            available.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name="featureValue" type="FeatureValueType"
minOccurs="0"
                    maxOccurs="unbounded">
        <xsd:annotation>
          <xsd:documentation>
            An optional list of feature values. If the request includes one
or
            more feature values of a featureFamily, the remaining values in
this
            family are ignored when generating the value permutations.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name="status" type="StatusType" minOccurs="1"
                           maxOccurs="unbounded">
        <xsd:annotation>
          <xsd:documentation>
            In case of success code will be "0", or In case of failure
error
            will be reported and code will be none "0".
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="ExecuteOperationRequest"
                           type="ExecuteOperationRequestType" />
  <xsd:element name="ExecuteOperationResponse"
                           type="ExecuteOperationResponseType" />
```

The following is the *request* example of the request body sent from Product Configurator for **DrillDiameter** of **1** and RPM of **100** which requests **CuttingSpeed**.

```
Request Body:
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<ExecuteOperationRequestType operationName="cuttingspeed" user="">
  <featureFamily description="" id="ID0000024D73E97D00" mandatory="1"
        modelFamily="0" name="RPM" namespace="000042" valueType="integer"/>
  <featureFamily description="" id="ID0000024D73E97F88" mandatory="1"
        modelFamily="0" name="DrillDiameter" namespace="000042"
```

```
        valueType="integer"/>
  <requestedFeatureFamily description="" id="ID0000024D73E98148"
mandatory="1"
        modelFamily="0"
  name="CuttingSpeed" namespace="000042" valueType="floatingPoint"/>
  <featureValue description="" id="ID0000024D033C6E08" name="100"
        namespaceRef="ID0000024D73E97D00"/>
  <featureValue description="" id="ID0000024D73E99168" name="1"
        namespaceRef="ID0000024D73E97F88"/>
</ExecuteOperationRequestType>
```

The following is the *response* example returned from the calculations Web service and consumed by Product Configurator with **CuttingSpeed** calculated as **26.1799387799**.

```
Response Body:
<?xml version='1.0' encoding='UTF-8'?>
<ExecuteOperationResponseType>
  <featureFamily description="" id="ID0000024D73E98148" mandatory="1"
        modelFamily="0" name="CuttingSpeed" namespace="000042"
        valueType="floatingPoint" />
  <featureValue description="" id="ID00000000049AE888" name="26.1799387799"
        namespaceRef="ID0000024D73E98148" />
</ExecuteOperationResponseType>
```

> Note:
>
> The response schema contains the **status code** because it is required in custom operations. In the current software release, the severity is not supported.

Add code in RESTful API and build the calculation server. Then, deploy it.

> Note:
>
> Use any standard framework for building and deploying RESTful APIs.

1. In RESTful APIs, handle the HTTP request as follows:

   - Check the HTTP request method, for example, GET or POST.
   - Verify the authentication sent in HTTP request.
   - Read the body from the HTTP request in the XML format.
   - Read the request XML and extract families and values.
   - Sample code is provided in C++ to parse XML and to write the response XML. This sample code is located at *<TeamcenterInstallation path>/TR/sample/Cfg0configurator*.

2. Run the calculations code.

3. Populate the response XML with request families and features.

4. Send the HTTP response.

   - Handle any errors in the calculations and appropriately send the HTTP response code.

Authenticating calculation Web service requests

Because calculations may contain proprietary algorithms, Teamcenter provides a mechanism to pass information in the request that may be used by the customer's calculation Web service to ensure the request is coming from Teamcenter and not from a malicious source intended to reverse-engineer the calculation service.

Product Configurator uses the JWT mechanism to authenticate the request. The system generates the JWT token and adds it in the HTTP request. The JWT token uses the secret key provided in the **ConfiguratorCalculationsSignerKeyPath** preference.

```
Authorization: Bearer <JWT Token>
```

> Note:
>
> To generate the JWT token, Product Configurator needs the **Signer Key**. An administrator needs to generate and provide the path of the **Signer Key** accessible in the Teamcenter server. The path must be set using the string **ConfiguratorCalculationsSignerKeyPath** preference. The preference value is `<Path of Signer Key>`.
>
> The **Signer Key** is a secret key. It must be stored in a secure location with restricted access.

Calculations web services may verify the JWT token which is present in the HTTP request. The verification must be performed using the **Verify Key** which is generated along with the **Signer Key**. The **Verify Key** is placed in the location where it can be accessed by the calculation Web service.

> Note:
>
> The **Signer Key** and the **Verify Key** are generated in the .PEM format.

Examples of the Verify Key and the Signer Key

- **Sample Signer Key (RSA private key)**

```
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAnzyis1ZjfNB0bBgKFMSvvkTtwlvBsaJq7S5wA+kzeVOVpVWw
kWdVha4s38XM/pa/yr47av7+z3VTmvDRyAHcaT92whREFpLv9cj5lTeJSibyr/Mr
m/YtjCZVWgaOYIhwrXwKLqPr/11inWsAkfIytvHWTxZYEcXLgAXFuUuaS3uF9gEi
NQwzGTU1v0FqkqTBr4B8nW3HCN47XUu0t8Y0e+lf4s4OxQawWD79J9/5d3Ry0vbV
3Am1FtGJiJvOwRsIfVChDpYStTcHTCMqtvWbV6L11BWkpzGXSW4Hv43qa+GSYOD2
QU68Mb59oSk2OB+BtOLpJofmbGEGgvmwyCI9MwIDAQABAoIBACiARq2wkltjtcjs
kFvZ7w1JAORHbEufEO1Eu27zOIlqbgyAcAl7q+/1bip4Z/x1IVES84/yTaM8p0go
amMhvgry/mS8vNi1BN2SAZEnb/7xSxbflb70bX9RHLJqKnp5GZe2jexw+wyXlwaM
+bclUCrh9e1ltH7IvUrRrQnFJfh+is1fRon9Co9Li0GwoN0x0byrrngU8Ak3Y6D9
D8GjQA4Elm94ST3izJv8iCOLSDBmzsPsXfcCUZfmTfZ5DbUDMbMxRnSo3nQeoKGC
0Lj9FkWcfmLcpGlSXTO+Ww1L7EGq+PT3NtRae1FZPwjddQ1/4V905kyQFLamAA5Y
lSpE2wkCgYEAy1OPLQcZt4NQnQzPz2SBJqQN2P5u3vXl+zNVKP8w4eBv0vWuJJF+
hkGNnSxXQrTkvDOIUddSKOzHHgSg4nY6K02ecyT0PPm/UZvtRpWrnBjcEVtHEJNp
bU9pLD5iZ0J9sbzPU/LxPmuAP2Bs8JmTn6aFRspFrP7W0s1Nmk2jsm0CgYEAyH0X
+jpoqxj4efZfkUrg5GbSEhf+dZglf0tTOA5bVg8IYwtmNk/pniLG/zI7c+GlTc9B
BwfMr59EzBq/eFMI7+LgXaVUsM/sS4Ry+yeK6SJx/otIMWtDfqxsLD8CPMCRvecC
2Pip4uSgrl0MOebl9XKp57GoaUWRWRHqwV4Y6h8CgYAZhI4mh4qZtnhKjY4TKDjx
QYufXSdLAi9v3FxmvchDwOgn4L+PRVdMwDNms2bsL0m5uPn104EzM6w1vzz1zwKz
5pTpPI0OjgWN13Tq8+PKvm/4Ga2MjgOgPWQkslulO/oMcXbPwWC3hcRdr9tcQtn9
Imf9n2spL/6EDFId+Hp/7QKBgAqlWdiXsWckdE1Fn91/NGHsc8syKvjjk1onDcw0
NvVi5vcba9oGdElJX3e9mxqUKMrw7msJJv1MX8LWyMQC5L6YNYHDfbPF1q5L4i8j
8mRex97UVokJQRRA452V2vCO6S5ETgpnad36de3MUxHgCOX3qL382Qx9/THVmbma
3YfRAoGAUxL/Eu5yvMK8SAt/dJK6FedngcM3JEFNplmtLYVLWhkIlNRGDwkg3I5K
y18Ae9n7dHVueyslrb6weq7dTkYDi3iOYRW8HRkIQh06wEdbxt0shTzAJvvCQfrB
jg/3747WSsf/zBTcHihTRBdAv6OmdhV4/dD5YBfLAkLrd+mX7iE=
-----END RSA PRIVATE KEY-----
```

- **Sample Verify Key (RSA public key)**

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAnzyis1ZjfNB0bBgKFMSv
vkTtwlvBsaJq7S5wA+kzeVOVpVWwkWdVha4s38XM/pa/yr47av7+z3VTmvDRyAHc
aT92whREFpLv9cj5lTeJSibyr/Mrm/YtjCZVWgaOYIhwrXwKLqPr/11inWsAkfIy
tvHWTxZYEcXLgAXFuUuaS3uF9gEiNQwzGTU1v0FqkqTBr4B8nW3HCN47XUu0t8Y0
e+lf4s4OxQawWD79J9/5d3Ry0vbV3Am1FtGJiJvOwRsIfVChDpYStTcHTCMqtvWb
V6L11BWkpzGXSW4Hv43qa+GSYOD2QU68Mb59oSk2OB+BtOLpJofmbGEGgvmwyCI9
MwIDAQAB
-----END PUBLIC KEY-----
```

Using **OpenSSL tool/API**, you can generate **RSA Public/Private Keys** that are used as the **Verify Key** and the **Signer Key**, respectively.

Your administrator can use any library or tools available over the internet to **generate the Signer Key and the Verify Key in .PEM format**.

Creating business objects for calculations

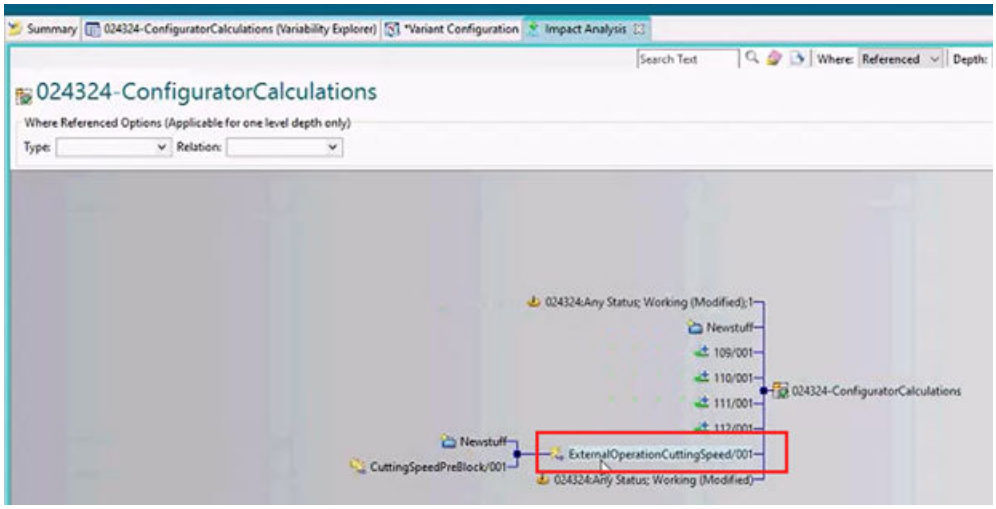
To use external calculations in Product Configurator, you create the following objects using TC or RAC ITK APIs:

- **External operation object**

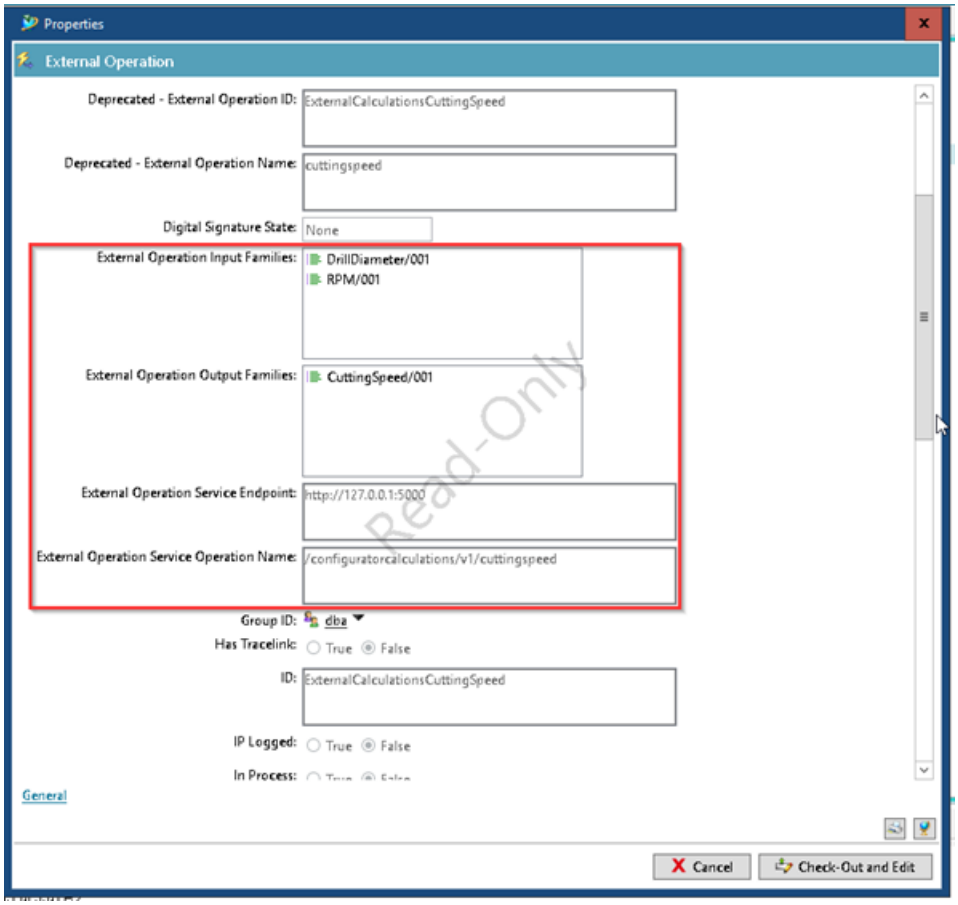
  For each external calculation, you create an **External Operation** object related to the corresponding configurator context. It specifies the service endpoint, the service operation name, the input families, and the output families. You can add more input and output families to the operation object to submit multiple families to the same external calculation or to request multiple output families from the same external calculation. The service operation name must be the same as the RESTful API. The service end point and the service operation name can be changed as needed.

  For example:

  Using **Impact Analysis**, you can view all objects referenced by the selected configurator context.



  The properties window below displays the external operation input and output families.

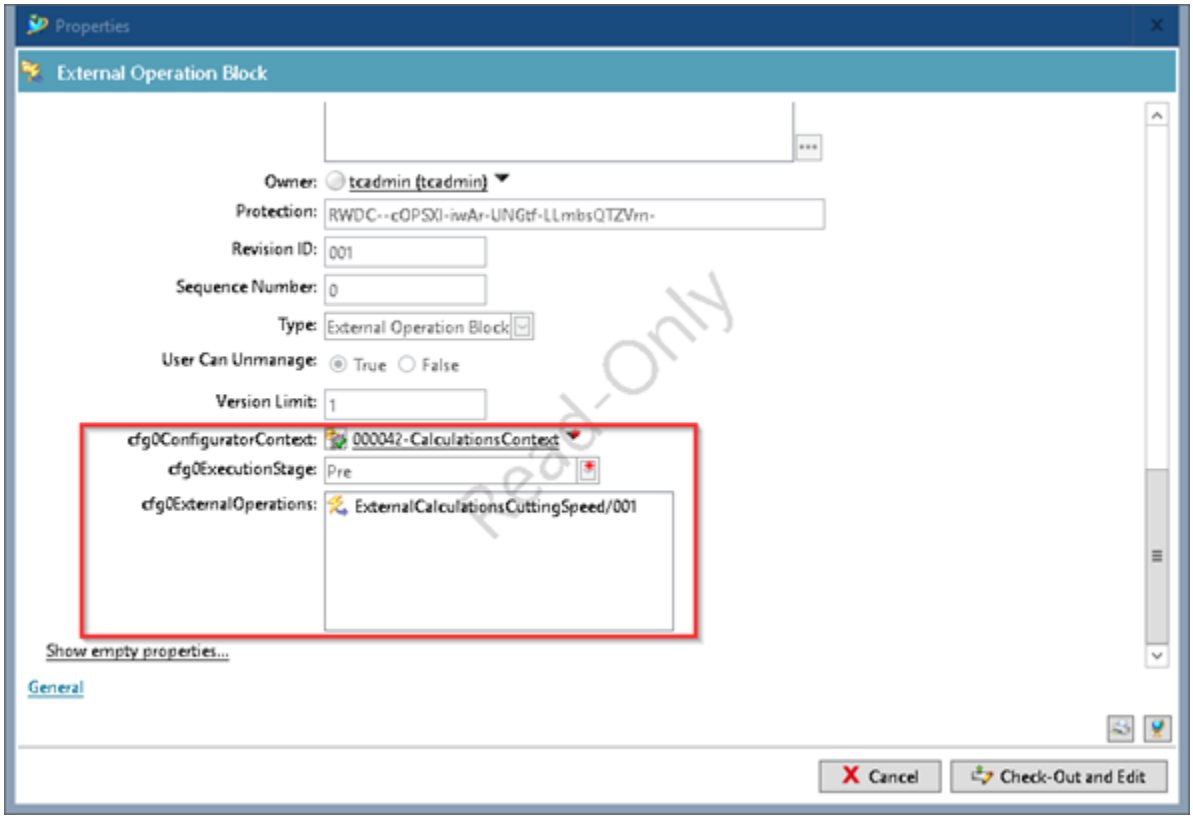


- **External operation block**

  External Operation Block is used to associate one or more **External Operation** objects with a PRE or POST calculation event for a given configurator context. This block object is related to the configurator context. It sets the applicable stage, either PRE or POST, and contains a list of **External Operation** objects. When called, Teamcenter dispatches each **External Operation** object in the sequence listed in the block.

  For example:

  As the properties window displays below, the following **External Operation Block** contains two external operation objects.



Calculations may be performed prior to the solve operation and after the solved operations. For a given configurator context, you can create only one external operation block object for the PRE phase (before the solve operations) and the POST phase (after the solve operations).

Performing calculations during solve in Product Configurator

In the current release, calculations are supported on the **Validate** and **Expand** solve commands. Calculations are performed before and after the solve.

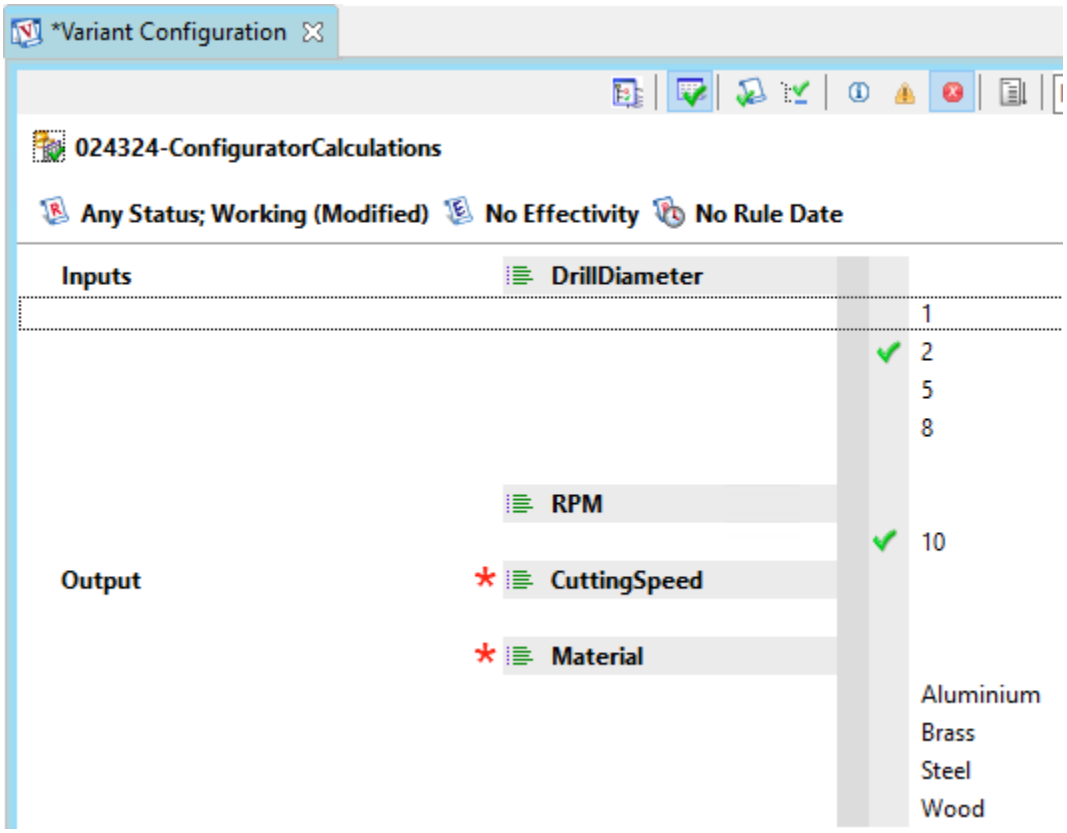
When you click **Validate** or **Expand** for variability configuration or BOM configuration, the system contacts the specified web services to perform the calculations.

> Note:
>
> Your administrator enables calculations by setting the **Cfg0EnableExternalOperationExecution** preference to true.

For example:

1. In Product Configurator, click **Open Variant Configuration view** on the toolbar to configure variants for the configurator context.

2. Type any values that may be required by PRE calculations.



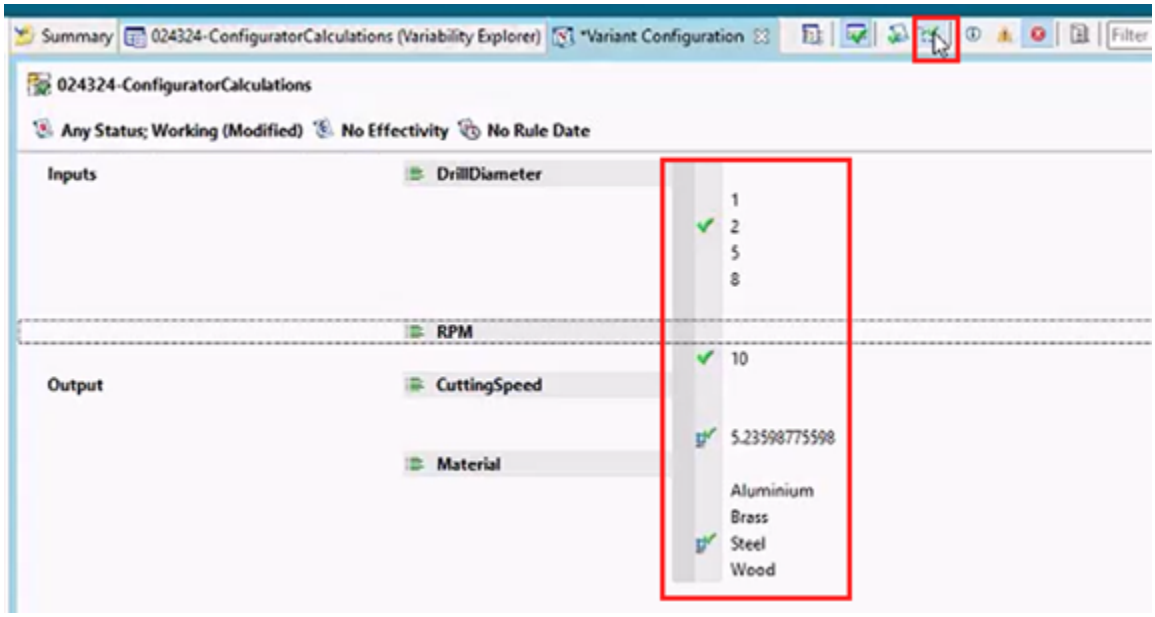
3. To view calculations results:

   - Click **Validate** to validate your input.
   - Click **Expand** to allow the system to consider all system rules to validate your input and assign features or precise values to the feature families using system knowledge.

4. Prior to solve in Product Configurator, the system locates an available **External Operation Block** for a given configurator context in the PRE phase. Once located, each external operation is executed for a corresponding block of configurator context. Results, given by a custom function, are updated as the input criteria.

   Modified and updated criteria is provided by the system to the solve operation.

5. After the solve executes again (POST phase), if a POST **External Operation Block** is present, the system performs calculations.



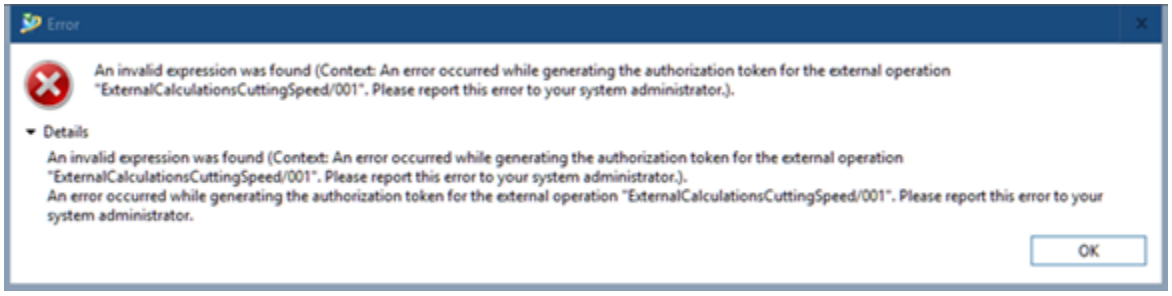
Troubleshooting errors during calculations in Product Configurator

If you experience problems with the calculation Web service, the system generates error messages. These errors must be reported to your system administrator.

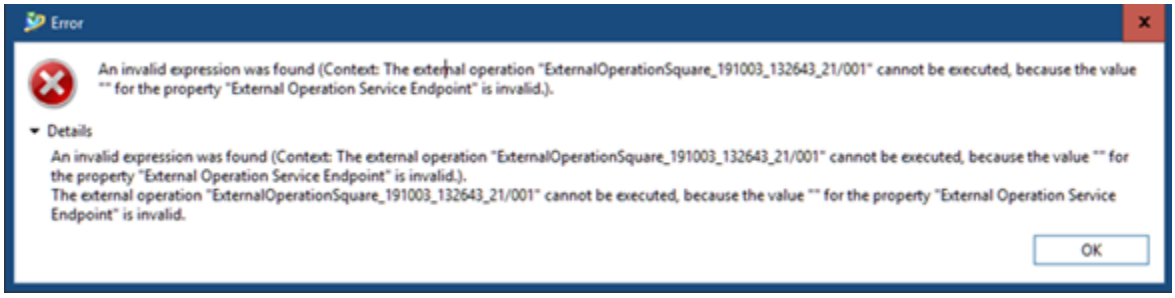
The following are examples of the errors reported by Teamcenter.

1. The calculation web service is not running or unable to connect.

2. Product Configurator fails to generate the JWT token due to an absence of key or an invalid key path.



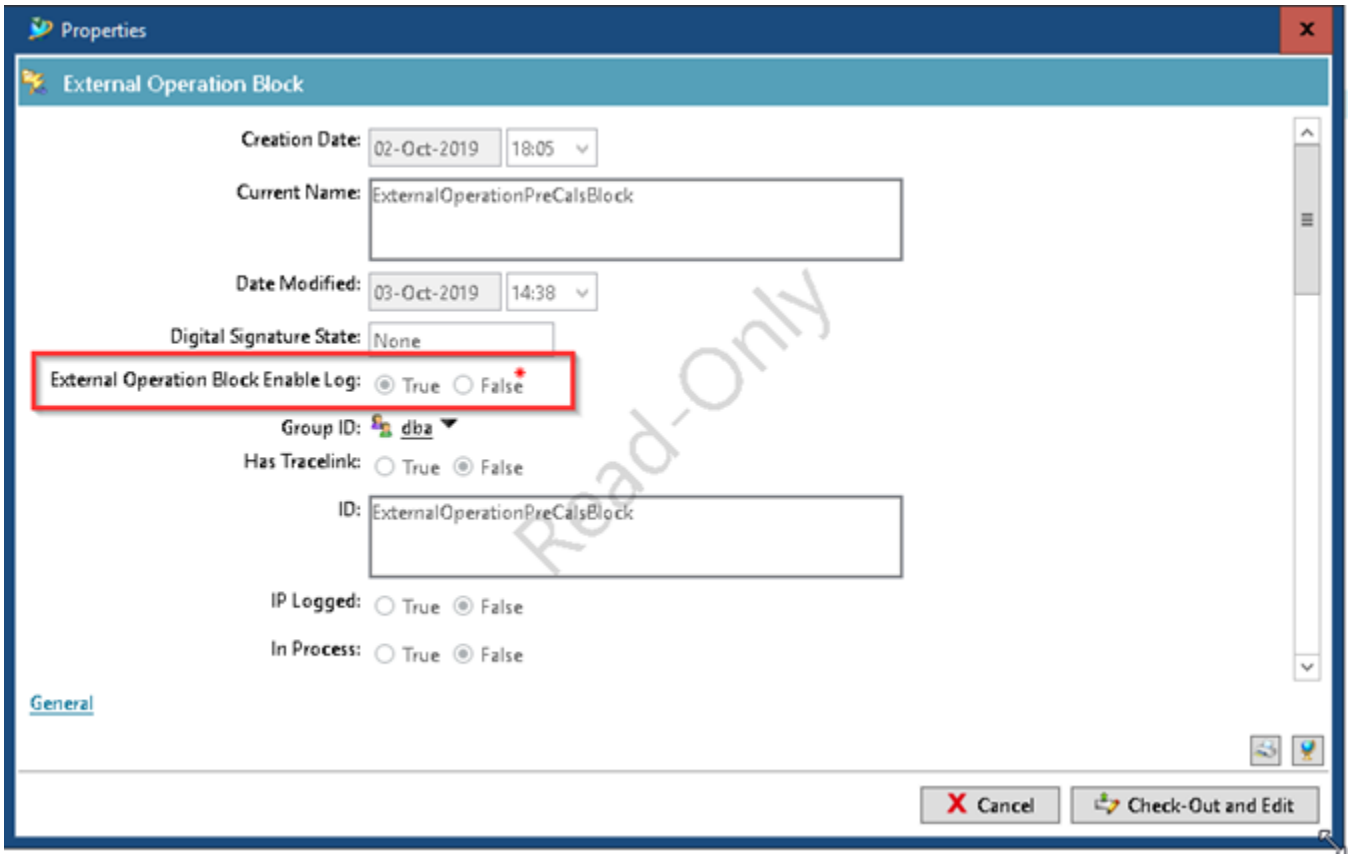
3. The calculation web service end point or service operation is empty or invalid.



Enable log files for the external operation block

Your administrator can enable the logging of the **External Operation Block** request and response. It is controlled by the **Enable Log on the External Operation Block** property.

If enabled, logs are written in the server **syslog** file.



Set Product Configuration preferences for Active Workspace

As an administrator, you can set the following Product Configuration site preferences or BMIDE properties for Active Workspace.

- Configure how product lines, product models, and features are displayed as thumbnails in the work area, as follows:

  - Set the **Cfg0AbsValue.cellProperties** site preference to define the list of properties displayed in each tile. Each entry in the list should be in the format *object_string , object_desc*. If the user wants to change the properties for a specific type of object, the administrator needs to create a preference with that type in the format *[ObjectType].cellProperties*. For example, **Cfg0AbsConfiguratorWSO.cellProperties**.

    Teamcenter allows you to use **Name** instead of **ID** as a primary business-relevant attribute in Product Configurator based on a global setting. For more information, see **Use Name or ID as the primary business attribute**.

  - Groups, families, and features are sorted according to the value of the **cfg0Sequence** property. If this property is same for two objects, the system sorts them alphanumerically. If thumbnails are enabled for a particular feature type, the system reads the *business_object*.**cellProperties** preference and checks if the **object_string** value is available. If so, it sorts according to the **object_string** value as determined from the preference.

    > Note:
    >
    > If you use Active Workspace 6.0 with Teamcenter 13.0 or 13.1, use the **cfg0ObjectId** or **object_name** instead of **object_string**.

- Set the default validation mode

  Create the **PCA_Default_Solve_Mode** preference and set it to **pca0Order** for **Order** or **pca0Overlay** for **Overlay** mode. This preference is not available by default.

  (Optional) Use the **PCA_solver_profiles** preference to create **custom solve profiles** in addition to the **Order** and **Overlay** modes.

Run the numeric feature report utility

When configurator administrators or designated users create a numeric feature, under a family type such as an integer or a float, the feature ID and the feature name must be the same. When they provide only the feature ID, the system automatically populates the feature name with the same ID value and vice versa.

As a user with administrative privileges, you can run the **report_numeric_feature_where_name_and_id_different** utility to create a report of the numeric

features where the names and IDs are different. For more information about running this utility, type *utility_name* **-h** on the Teamcenter command prompt.

Optimize configurator rules to improve solver performance

You can optimize configurator rules to improve solver performance by using the **constraint_optimizer** utility. When you run this utility, it simplifies the configurator rules in a given user session that includes the configurator context, revisions rules, and solver profile such as **Order Mode** or **Overlay Mode**.

For more information about running this utility, type **constraint_optimizer -h** on the Teamcenter command prompt.

Consider a simple configurator context as follows (example):

Model	
	M1
	M2
	M3
FamA (single select)	
	A1
	A2
	A3
FamB (single select)	
	B1
	B2
	B3

Configurator rules (example):

R1: (M1 OR M2) Includes A1

R2: M3 Includes A3

R3: A1 Excludes (B1 AND NOT(M2))

R4: A2 Excludes B2

R5: A3 Excludes B3

There are five configurator rules for this example. However, the list of relevant constraints can be trimmed if the model is known in advance before executing the solve.

Input selection for model family	Relevant constraints
M1	**R1** is simplified to **M1 Includes A1** **R2** is optimized away **R3** is simplified to **A1 Excludes B1** **R4** is optimized away **R5** is optimized away
M2	**R1** is simplified to **M2 Includes A1** **R2** is optimized away **R3** is optimized since it always evaluates to **True** **R4** is optimized away **R5** is optimized away
M3	**R1** is optimized away **R2** is unchanged but relevant **R3** is optimized away **R4** is optimized away **R5** is unchanged but relevant

After running the constraint optimization utility, a constraint can have three states:

- Constraint remains active but was simplified.
- Constraint is detected as inactive and is temporarily removed from the configuration snapshot, if and only if it collapses to a **Tautology** given the input configuration for which the rule set is optimized.
- Constraint remains active in its original form.

There are many real world scenarios where for a given choice of configuration criteria, the relevant constraint list can be reduced to a considerably smaller size than the complete constraint list. This trimming down of the constraint list can lead to faster solve execution and hence result in considerable performance improvement during solve related operations.

Optimization based on inclusion criteria

To perform the optimization based on the models, you can include all the models in the inclusion criteria file.

inclusive_criteria.txt example:

[Namespace]Model=M1

[Namespace]Model=M2

[Namespace]Model=M2

Run the following command on the Teamcenter command prompt:

constraint_optimizer -u=*Tc-admin-user* **-p=***Password* **-contextItem=***Product_Item1 fnd0VariantNamespace property* **-revision_rule_name=***Any Status; Working* **-rule_date=***3-Jan-2022 10:30* **-icf=***D:/inclusive_criteria.txt* **-spf=***D:/solver_profiles.json*

Where **-icf=** specifies the inclusion criteria to perform constraint optimization and **-spf=** specifies the JSON file containing the list of solve operation profiles for which configuration optimization is computed.

Solve operation profile JSON file example:

```
{
    "_comment": "This is a sample provided for customer reference.",
    "description": "The solver profile file with a single solver
profile,
    that is, overlay profile. This is used as an input for constraint
    optimization utility for generating optimization.",
    "solveProfiles": [
        [
            {
                "key": "CFG_PROFILE_applyConfigConstraints",
                "value": true
            },
            {
                "key": "CFG_PROFILE_applyDefaults",
                "value": false
            },
            {
                "key":
"CFG_PROFILE_considerLowSeverityRulesForExpansion",
                "value": false
            },
            {
                "key": "CFG_PROFILE_minErrorSeverity",
                "value": "Error"
            },
            {
                "key": "CFG_PROFILE_minReportSeverity",
                "value": "Error"
            },
            {
                "key": "CFG_PROFILE_preferEmptyValueForDiscretionary",
```

```
                "value": false
            },
            {
                "key": "CFG_PROFILE_violationComputationTimeout",
                "value": 4
            }
        ]
    ]
}
```

Optimization based on reference families

To optimize every feature value in **FamA**, instead of providing the inclusion criteria containing **A1**, **A2**, and **A3**, you can provide the reference family name.

reference_family.txt example:

[Namespace]FamA

This is instead of providing all the feature values in the inclusion criteria (**inclusive_criteria.txt**) as follows:

[Namespace]FamA=A1

[Namespace]FamA=A2

[Namespace]FamA=A3

If you specify multiple family names in **reference_family.txt**, the system takes the **Cartesian product** or the set of all possible ordered combinations consisting of one member from each of these sets.

reference_family.txt example for multiple families:

[Namespace]FamA

[Namespace]FamB

This is similar to providing **inclusive_criteria.txt** as follows:

[Namespace]FamA=A1 & [Namespace]FamB=B1

[Namespace]FamA=A1 & [Namespace]FamB=B2

[Namespace]FamA=A1 & [Namespace]FamB=B3

[Namespace]FamA=A2 & [Namespace]FamB=B1

[Namespace]FamA=A2 & [Namespace]FamB=B2

[Namespace]FamA=A2 & [Namespace]FamB=B3

[Namespace]FamA=A3 & [Namespace]FamB=B1

[Namespace]FamA=A3 & [Namespace]FamB=B2

[Namespace]FamA=A3 & [Namespace]FamB=B3

Run the following command on the Teamcenter command prompt:

-u=*Tc-admin-user* **-p=***Password* **-contextItem=***Product_Item1 fnd0VariantNamespace property* **-revision_rule_name=***Any Status; Working* **-rule_date=***3-Jan-2022 10:30* **-rff=***D:/reference_family.txt* **-relation=***IMAN_reference* **-spf=***D:/solver_profiles.json*

Where

-rff= specifies the file containing a list of reference families for optimization.

-spf= specifies the **JSON file** containing the list of solve operation profiles for which configuration optimization is computed.

-relation= specifies the generic relationship management (GRM) relationship type name with which the optimized configuration snapshot should be attached to the configurator context item, for example, *IMAN_reference*.

If a GRM relationship exists between the configurator context item and the configuration snapshot, the attached rule set is used whenever users specify the system default rule date (see **Cfg0RuleDateOffset** preference) even if the **Configured As Of Date** date of the rule set does not match the system default rule date. That is, the presence of a GRM relation overrides the regular date based search mechanism to ensure that the users get the benefit of an optimized rule set.

Depending on the way the revision rule is defined, the system may display warnings while opening the **Configuration** view if the configurator objects were created, deleted, or changed in the period between the **Configured As Of Date** date and the current system default rule date. In such a case it might be required to update the **Cfg0RuleDateOffset** preference so that the system default rule date and the **Configured As Of Date** date match.

Optimization based on exclusion criteria

To perform optimization based on exclusion criteria, you can exclude certain features from different families. For example, you do not want to optimize **A3 & B1** and **A2 & B2**. In such a case you can provide the reference families and the exclusion criteria.

reference_family.txt example:

[Namespace]FamA

[Namespace]FamB

exclusive_criteria.txt example:

[Namespace]FamA=A3 & [Namespace]FamB=B1

[Namespace]FamA=A2 & [Namespace]FamB=B2

This is similar to providing **inclusive_criteria.txt** as follows:

[Namespace]FamA=A1 & [Namespace]FamB=B1

[Namespace]FamA=A1 & [Namespace]FamB=B2

[Namespace]FamA=A1 & [Namespace]FamB=B3

[Namespace]FamA=A2 & [Namespace]FamB=B1

[Namespace]FamA=A2 & [Namespace]FamB=B3

[Namespace]FamA=A3 & [Namespace]FamB=B2

[Namespace]FamA=A3 & [Namespace]FamB=B3

Run the following command on the Teamcenter command prompt:

-u=*Tc-admin-user* **-p=***Password* **-contextItem=***Product_Item1 fnd0VariantNamespace property* **-revision_rule_name=***Any Status; Working* **-rule_date=***3-Jan-2022 10:30* **-rff=***D:/reference_family.txt* **-ecf=***D:/exclusive_criteria.txt* **-relation=***IMAN_reference* **-spf=***D:/solver_profiles.json*

Where

-rff= specifies the file containing a list of reference families for optimization.

-ecf= specifies the exclusion criteria to be excluded during constraint optimizations from all build combinations of referenced families.

-spf= specifies the **JSON file** containing the list of solve operation profiles for which configuration optimization is computed.

-relation= specifies the generic relationship management (GRM) relationship type name with which the optimized configuration snapshot should be attached to the configurator context item, for example, *IMAN_reference*.

If a GRM relationship exists between the configurator context item and the configuration snapshot, the attached rule set is used whenever users specify the system default rule date (see **Cfg0RuleDateOffset** preference) even if the **Configured As Of Date** date of the rule set does not match the system default rule date. That is, the presence of a GRM relation overrides the regular date based search mechanism to ensure that the users get the benefit of an optimized rule set.

Depending on the way the revision rule is defined, the system may display warnings while opening the **Configuration** view if the configurator objects were created, deleted, or changed in the period between the **Configured As Of Date** date and the current system default rule date. In such a case it might be required to update the **Cfg0RuleDateOffset** preference so that the system default rule date and the **Configured As Of Date** date match.

Using Product Configurator utilities

Utility name	Description
cfg0_add_solveprofile_on_variantrule	Sets solve profile on variant rule objects.
cfg0_gen_enforced_cond	Attaches an enforced condition to all constraint rules that do not have any existing enforced conditions attached.
cfg0_ignore_constraint_on_vrule	Searches variant rules for configurator contexts, SVR names, or SVR IDs as input and updates boolean values for applying configurator constraints on SVRs. Valid values are **true** and **false**, and **false** is the default value.
cfg0_install_am_rule	(System administration utility) Adds an Access Manager rule entry to the Access Manager tree. This rule ensures that the rule compiled dataset is not readable by anyone except DBA members.
cfg0_stamp_completeness_on_vrule	Searches variant rules for configurator contexts, SVR names, and SVR IDs and stamps completeness state on SVRs if the SVR is valid and complete.
cfg0_update_ruledate_translation_key	Searches variant rules for configurator contexts, SVR names, and SVR IDs as inputs and updates the rule date translation key value on SVRs.

For more information about running these utilities, type *utility_name*-**h** on the Teamcenter command prompt.

Generate the latest snapshot for modular configurations

In a modular configuration, typically, configuration contexts are hierarchically represented using configuration modules. The modular hierarchy can potentially include a large number of configuration modules. In order to perform Variant Configuration in the context of such modular hierarchies, a single consolidated snapshot is generated that includes all the configuration context data and constraints across the given hierarchy depth. Generating such a snapshot online is time consuming and thus the

Teamcenter web client has the prerequisite that the compiled rule set for the given depth is already available for validation, expansion, and solve use cases.

As an administrator, you can run the **generate_compiled_ruleset_snapshot** utility as a cron job on the configuration context that has a modular hierarchy to always get the latest snapshot. This results in a quicker solve.

For more information, see *Improving the system response time for the configuration requests using a snapshot mechanism*.

The parameters required for running this utility are the configuration context ID, revision rule configurations (revision rule, rule date) and hierarchy depth. A depth greater than **1** is used to include the module hierarchy.

For more information about this utility, type **generate_compiled_ruleset_snapshot -h** on the Teamcenter command prompt.

Create advanced search queries for variant rules

You can create custom advanced search queries using **Query Builder** to increase the efficiency of searches by narrowing the criteria and expediting the results.

You can create the search query by accessing the **Query Builder** tile on the home page or by using **Save As** on an existing query.

After creating a custom saved query for variant rules, you can edit the **PCA_VariantRule_SavedQueries** site preference to add the saved query to the **Search by** option in the **Load Variants** dialog box. The query should be based on the **VariantRule** search type in the dialog box. The query should always be associated with the **VariantRule** business object or its subtypes.

The **Variant Rules (Saved Query)** is available by default in the **Load Variants** dialog box. It contains fields such as **Name**, **Description**, **Revision ID**, and **ID**. If any of these fields are removed from the saved query, they become unavailable in the dialog box.

Disable authoring of free form rules

Configurator administrators can create complex free-form SMT-based rules using the SMT Lib scripting language.

By default, the **Cfg0EnableFreeFormRuleSupport** site preference is set to **true**. You can disable authoring of free form rules by setting this preference to **false**.

Enable the Modules tab in Active Workspace

You can enable the **Modules** tab in Active Workspace to allow users to browse the configurator modules by setting the **PCA_Display_Modules** site preference to **true**. By default, it is set to **false**.

Allow users to create variant rules or variant criteria

As the administrator, you can control whether users create variant rules or variant criteria using the **Cfg0CreateVariantRuleType** site preference. The preference can be set to either **VariantRule** or **Cfg0VariantCriteria**, which allows end users to create either variant rules or variant criteria, respectively after selecting features in a configurator context. By default, it is set to **Cfg0VariantCriteria**. Siemens Digital Industries Software does not recommend using both variant criteria and variant rules at the same time. If you have started using variant criteria, do not switch back to variant rules.

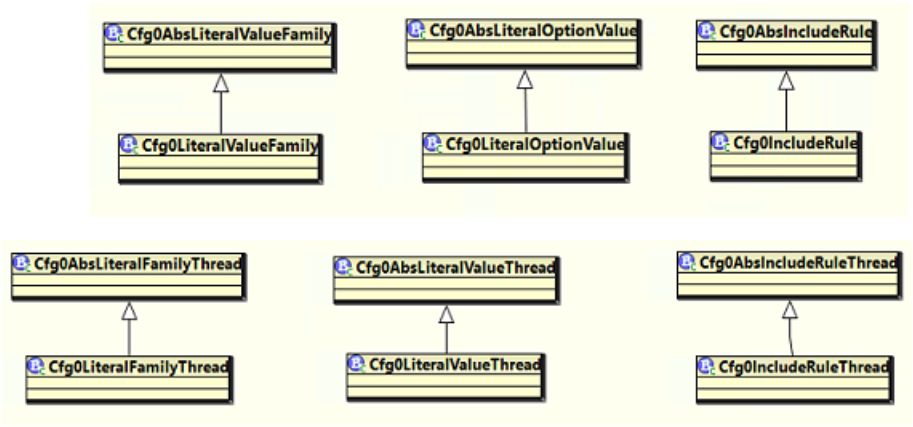
Allow users to save configuration changes to the current structure or configurator context

You can set the **PCA_attach_variant_to** site preference to allow users to users to save configuration changes to the current structure or the configurator context. By default, it is set to **Context** for configurator context.

7. Customizing Product Configurator

About the Product Configurator data model

In Product Configurator, the data model for business objects distinguishes between user-facing objects that are instantiable and their abstract parent classes that provide attributes, properties and core functionality.



Every functional Product Configurator type, such as Feature (`Cfg0LiteralOptionValue`), Family (`Cfg0LiteralValueFamily`), or Inclusion Rule, contains a pair of the abstract class and its instantiable subtype.

The generic and functional behavior of objects in Product Configurator is implemented and managed as the set of abstract classes. These abstract classes are inheritable. Users can subtype the appropriate abstract classes, if required. The instantiable types inherit the behavior from respective abstract parent classes, and they do not extend any additional functionality. By default, the instantiable classes are leaf types that cannot be further subtyped. Siemens Digital Industries Software recommends not to customize the default types.

> Note:
>
> Using Business Modeler IDE, you can create subtypes of the instantiable types. However, the deployment of such a data model will fail.

Create, modify, update, or delete configurator objects by using recommended ITKs

The recommended ITK APIs to load, unload, delete, and save objects in the Product Configurator data model, for example, **AOM_save_with_extensions**, are mentioned in **Files** > **tccore** > **aom.h (code)** in *Integration Toolkit Function Reference*.

These APIs execute all extensions that are defined for the corresponding base action. These extension must be executed to maintain data consistency for objects in the Product Configurator data model.

When it is certain that these extension are not required, you can use appropriate APIs in the modules, for example, **AOM_save_without_extensions**. This can be useful for some scenarios, such as updating the last modified date update or intermediate save operations for member objects during complex operations.

Siemens Digital Industries Software recommends that you use bulk APIs, for example, **AOM_bulk_save_instances**, where appropriate.

Product Configurator business type constants

Constant	Purpose	Default value
Cfg0EnableEffectivityConfigurableBehavior	Specifies if the type supports the effectivity functionality. By default, the following object types support effectivity: • Feature • Product Model • Summary Model • Feature Package • Feature Summary • Inclusion Rule • Default Rule • Exclusion Rule • Availability Rule • Configurator Allocation Siemens Digital Industries Software recommends to change the type constant	False

Constant	Purpose	Default value
	only for above types or their subtypes.	
Cfg0ThreadType	Specifies the type of the thread class to be used when a new instance is created. Caution: You must populate a valid instantiable thread type name for all instantiable revision types. The instance creation fails if this type of constant value is left empty.	Empty
Cfg0DefaultValueType	Specifies which value type or subtypes are allowed for the family type. This type constant is applicable for family types. Caution: You must populate a valid instantiable thread type name for all instantiable family types. The instance creation of the value for the family fails if this type of constant value is left empty.	Empty

Product Configurator business types and the system checks they perform

Product Configurator provides implicit checks to ensure the consistency of the system. These checks cannot be changed or extended through customization.

Type	Description	System check
Family and its values	Allows to keep consistency between a family and its value. The system ensures	List of family types and its allowed value

Type	Description	System check
	that only certain value types are allowed according to the type of the family.	types and subtypes, as follows: • Family – Feature • Summary Family – Feature Summary • Package Family – Feature Package • Model Family – Product Model • Summary Model Family – Summary Model • Product Line Family – Product Line
Product Configurator object type and its thread	In addition to the **Cfg0ThreadType** type constant, the system ensures that appropriate thread types are used for the Product Configurator types. For example, system will not allow to use a value thread type used for the family type instance.	The Product Configurator object type must use the corresponding thread type in the thread type hierarchy.

Set preferences to manage custom Product Configurator types

Type	Scope	Purpose	Default value
Cfg0CompoundOptionValueMemberTypes	Site	Specifies the type of value allowed for the compound value, such as package, summary, to be allowed as members. This is a multi-	**Cfg0SummaryOptionValue:: Cfg0LiteralOptionValue Cfg0PackageOptionValue:: Cfg0LiteralOptionValue**

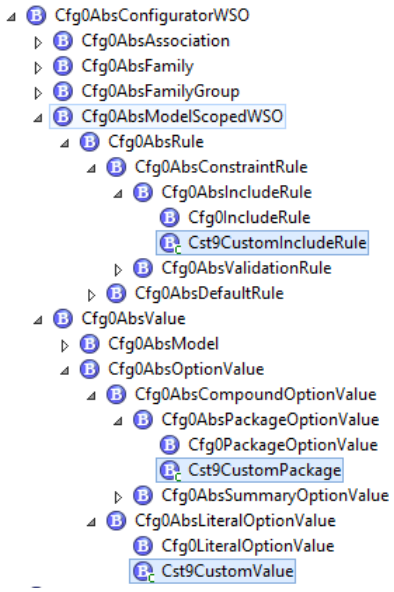
Type	Scope	Purpose	Default value
		value preference. The preference is defined as **<Compound Value type>:<Allowed member value type>**	
Cfg0OptionsDataSortProperty	User	Specifies the column to be used for sorting the Product Configurator application views. The property internal name must be specified as the value.	**cfg0Sequence**
Cfg0DefaultOptionFamilyNamespace	User	Specifies the default family namespace property value for new families created in the **Variability Explorer** view in Product Configurator.	**Teamcenter**
Cfg0AvailabilityMatrixLoadCount	User	Specifies the maximum number of families to be loaded in the **Availability Matrix** view as pagination strategy.	**5**

Examples to create a custom package, feature, and inclusion rule

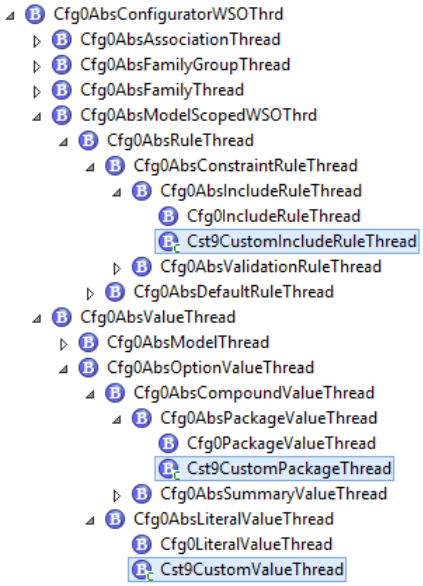
In the following example, you create custom types for a package, a feature, and an inclusion rule.

1. Create a Business Modeler IDE project with appropriate solutions.

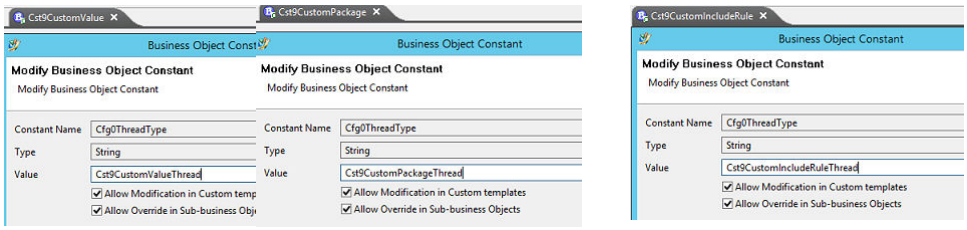
2. Create custom Package, Value, and Inclusion rule revisable types.



3. Create custom Package Thread, Value Thread, and Inclusion rule thread types.



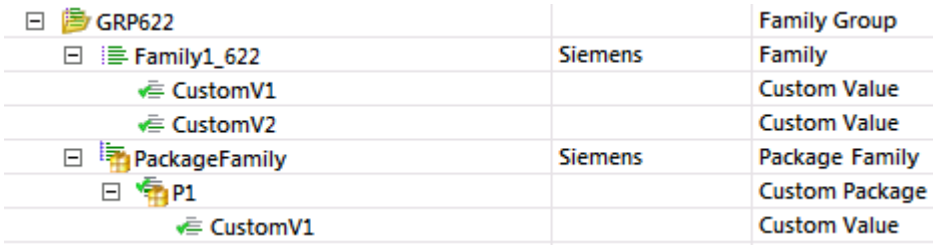
4. Associate the revisable types and respective thread types.



5. Deploy the data model.

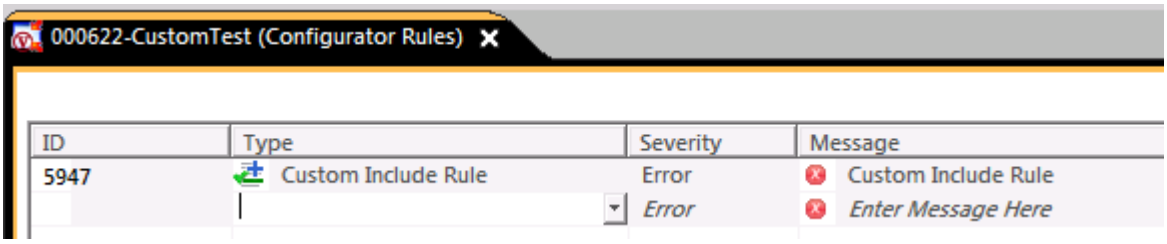
6. In Product Configurator, open the **Variability Explorer** view.

   Create custom values for your feature family. You can also create custom packages and custom values for your package families.



7. Open the **Configurator rules** view.

   Create a custom include rule.



Allow users to revise custom types by editing the project.xml file

1. To allow users to revise custom types, add the following text in the project.xml file.

```
   <OperationInputType typeName="Cst9CustomValueRevI"
parentTypeName="Cfg0AbsLiteralOptionValueRevI"
       typeClassName="Cst9CustomValueRevI" description="" isAbstract="false"
       artifactName="Cst9CustomValueRevI"/>
   <TcTypeConstantAttach constantName="ReviseInput" typeName="Cst9CustomValue"
       value="Cst9CustomValueRevI"/>
```

2. Deploy your project.xml.

   If you fail to add the above text in the step 1, users may see the following error when revising the custom type value in Product Configurator.

Create custom objects required for Product Configurator by using BMIDE

Configure Product Configurator using the Business Modeler IDE

Use the Business Modeler IDE to create custom objects used by the Product Configurator application.

Note:

Before working with Product Configurator objects, you must install the following templates to your project:

- Build Conditions (**bcs0buildconditions** file)
- Configured Search Framework (**srh0apsconfiguredsearch** file)
- Teamcenter Configurator (**cfg0configurator** file)

Product Configurator rules business objects

You can create children of the following Product Configurator rules business objects:

- **Cfg0AbsIncludeRule**

  Represents the abstract inclusion rule.

- **Cfg0IncludeRule**

  Defines the standard inclusion rule.

- **Cfg0AbsExcludeRule**

  Represents the abstract exclusion rule.

- **Cfg0ExcludeRule**

  Defines the standard exclusion rule.

- **Cfg0AbsFeasibilityRule**

  Represents the abstract feasibility rule.

- **Cfg0FeasibilityRule**

  Defines the standard feasibility rule.

- **Cfg0AbsDefaultRule**

  Represents the abstract default configuration rule.

- **Cfg0DefaultRule**

  Defines the standard default configuration rule.

If you are creating a child of an abstract rule business object, you must provide a value for the **Cfg0ThreadType** business object constant on the new business object to support creation of a child.

Add custom intents for Product Configurator

The **cfg0Intents** property on the **Cfg0AbsConfiguratorWSO** business object specifies the object intent. The **cfg0Intents** property obtains its values from the **Cfg0ObjectIntentions** list of values.

If you want to add custom intents, add them to the **Cfg0ObjectIntentions** list of values.

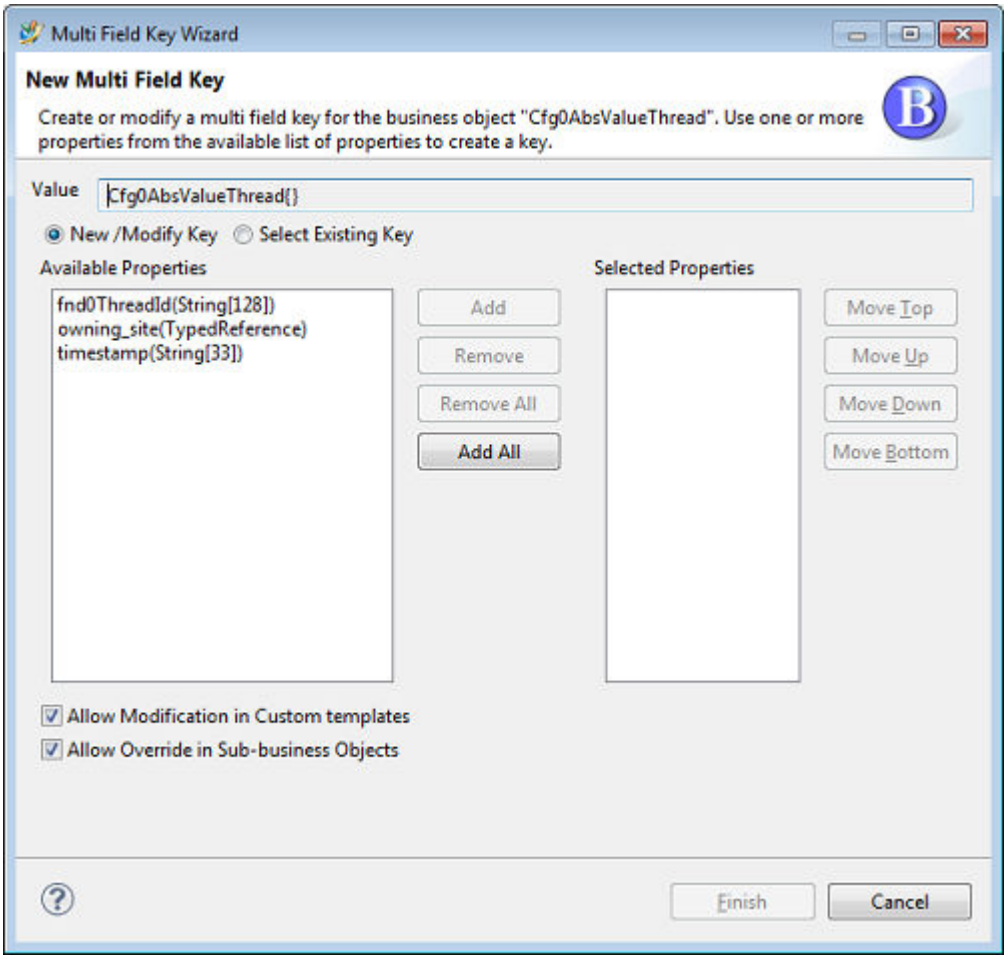
Set global uniqueness for a Product Configurator value

By default within Product Configurator, a family is unique within a namespace and a value is unique within the family. You can make this restriction more stringent by making a value unique everywhere through the use of multifield keys.

1. Open the **Cfg0AbsValueThread** business object.

2. On the **Business Object Constants** tab, select the **MultifieldKey** constant, and click the **Edit** button.

   The **New Multi Field Key** dialog box is displayed.



3. Select the **fnd0ThreadID** property and click the **Add** button to add it to the **Selected Properties** box.

4. Click **Finish**.

5. Save your changes and install the template.

Set context uniqueness for a Product Configurator value

By default within Product Configurator, a family is unique within a namespace and a value is unique within the family. You can make this restriction more stringent by setting a value to be unique within a context. You must add new extensions and author the appropriate code so that there are not conflicting instances of a value in a given product context.

1. Add a new post action extension on the save method of the **Cfg0AbsAllocation** business object.

2. From the **va_list** input supplied to this method, get the option value instance on which this post action got triggered.

3. Get the **fnd0ThreadId** property of the option value instance.

4. Using the configurator perspective object, get the option values that are allocated to the current context.

5. Check if there is any already allocated option value instance that has the same ID.

   If there isn't any such instance, set the return call so that the instance gets saved. If there is such a conflict, return the corresponding **IFail** token so that the save fails.

Define naming rules to customize configurator objects

You can define naming rules to customize the Product Configurator object IDs.

Siemens Digital Industries Software recommends that you set up naming rule guidelines for the required objects. The system administrator can attach multiple naming rule schemes to configurator object threads based on business needs.

Naming rules can be attached to the Product Configurator features, but use these guidelines for the Boolean data type:

- You should attach the same naming rule to both the Boolean family and Boolean value. Only attaching the naming rule to the Boolean value generates an error message when only Boolean value is created.

- If you must attach different naming rules for Boolean family and Boolean value, customize the Boolean family type and use it in Product Configurator.

Key points about the Product Configurator object IDs are as follows:

- Many Product Configurator objects have uniqueness constraints that involve the object ID.

- By default, Product Configurator does not enforce any naming rule patterns nor restrict the object IDs to any specific pattern.

  The system administrator sets up the required naming rules for configurator objects to ensure that assigned configurator object IDs are unique when users:

  - Create *new* configurator objects by either defining their own or using the system generated object IDs.

  - Import data.

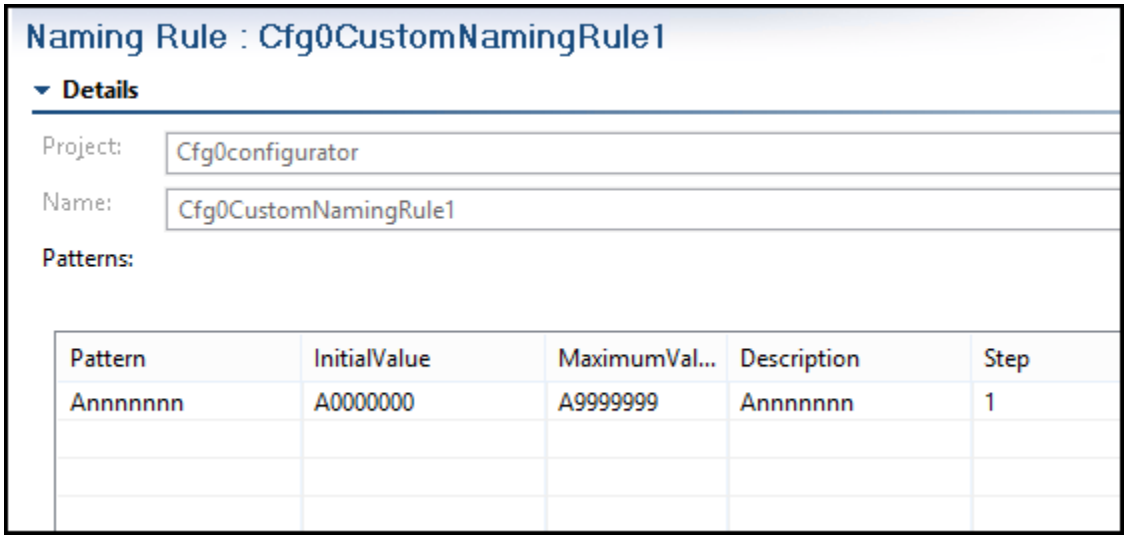
  When creating new objects, users have to either specify an ID or leave it blank. When users specify an ID, Teamcenter assigns the user-specified ID. When users leave the ID input field blank, the system checks whether naming rules are attached to this field. If the system does not find any naming rules

associated with that object, the system uses an internal number generator to generate a new object ID.

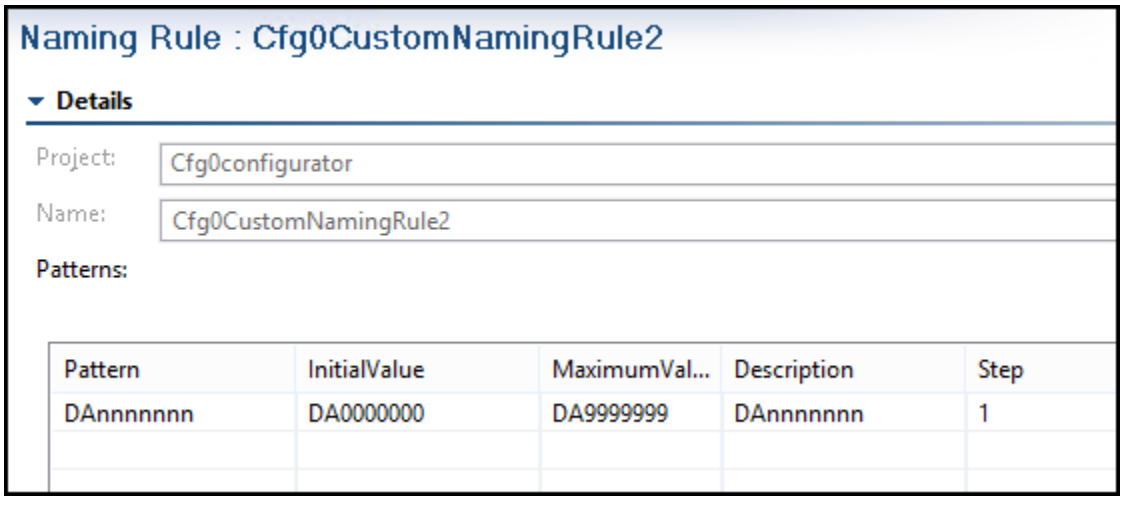
> Caution:
>
> If the system administrator does not provide an explicit naming rule to control the generation of an object ID for a newly created configurator object, the system generates the ID based on the default pattern.

Example of naming rules to customize configurator objects

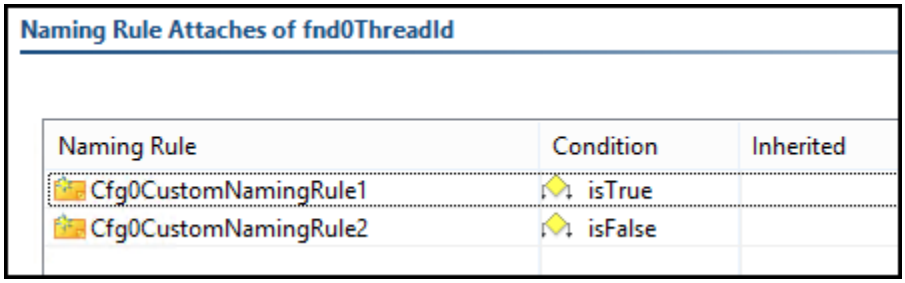
Example of a custom naming rule for creating new business objects using the rich client.



Example of a custom naming rule for importing data from a different site.



Both rules must be attached to **Cfg0AbsConfiguratorWSOThread.fnd0ThreadId** as follows:



Create naming rules for configurator objects

You can create naming rules to customize the naming of configurator objects. By default, no naming rules are available. However, you can create a naming rule with a proper sequence to identify different configurator objects. For example, you can create an inclusion rule that starts with *INC0001*. This number counter can keep incrementing till *INC9999*.

1. **Create a BMIDE template project**.
2. **Create a new naming rule**.
3. **Add the naming rule you created by modifying the Cfg0ABsConfiguratorWSOThrd business object**.
4. **Generate the software package to apply the changes**.
5. **Apply the changes using TEM**.
6. **Verify the changes in Active Workspace**.

Procedure

1. Create a BMIDE template project.

    a. Launch Business Modeler IDE (BMIDE).

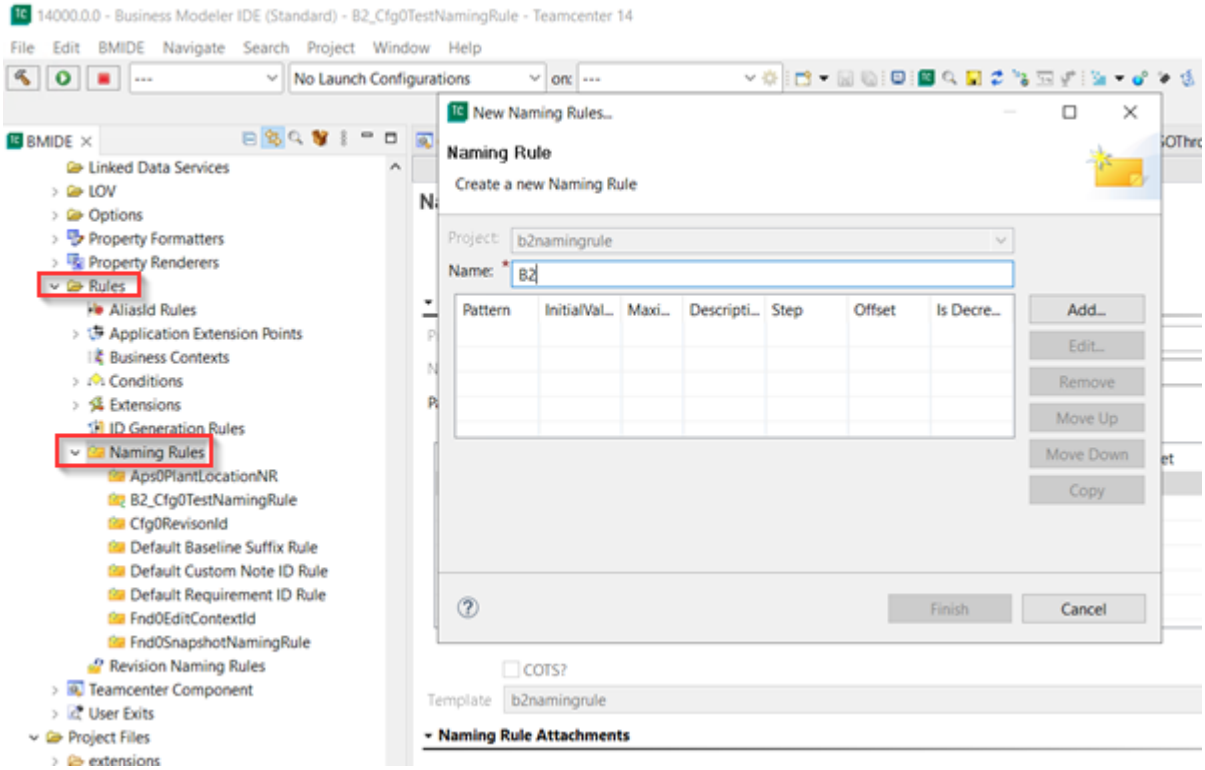
    b. Create a new BMIDE template project, for example, **b2namingrule**.

    c. In **Dependent Templates**, search for **cfg** and select **Teamcenter Configurator**.

    d. Search and select all other dependent templates, and click **Finish**.

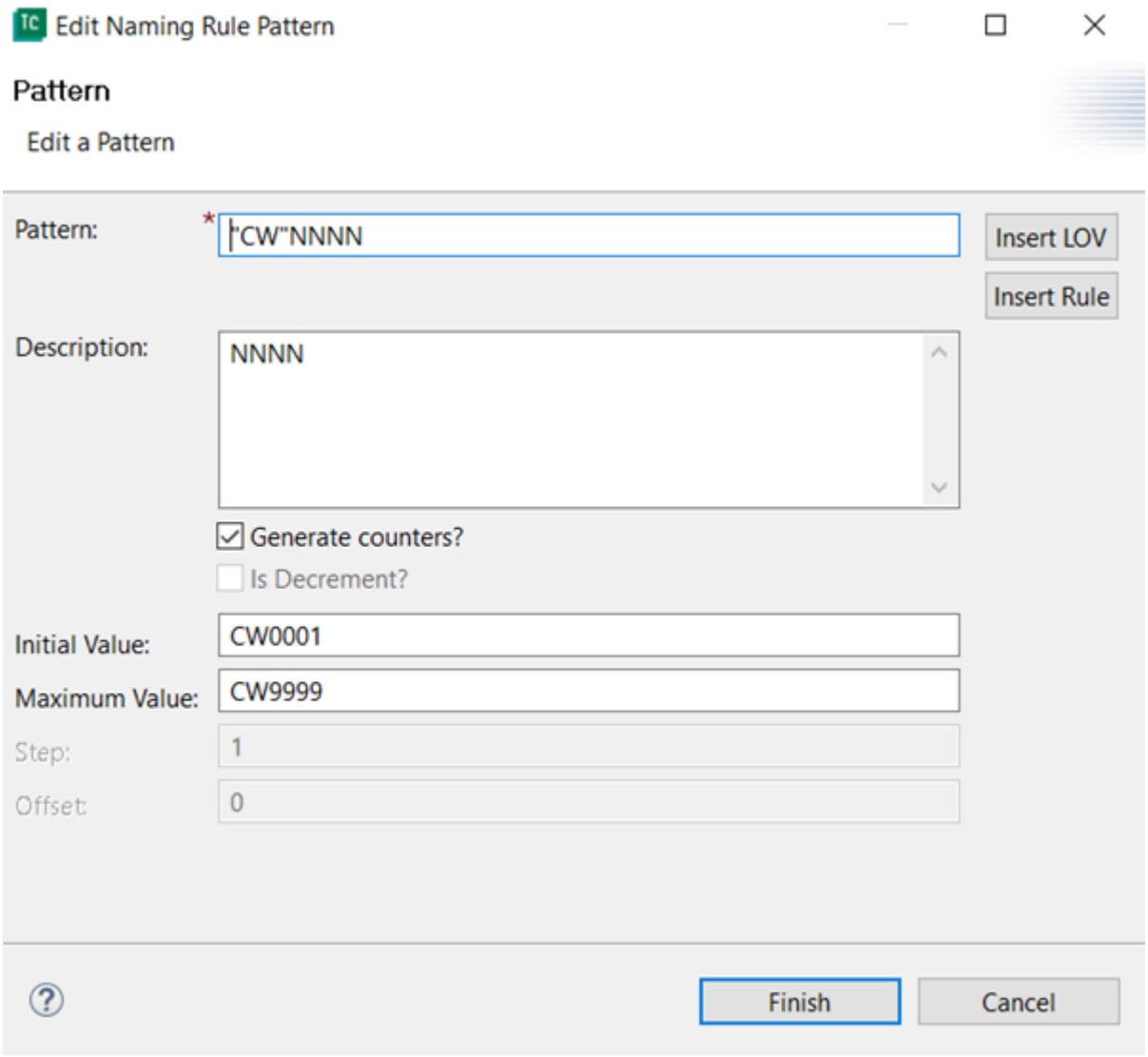
2. Create a new naming rule.

    a. In Business Modeler IDE, select **Extensions\Rules\Naming Rules**, right-click, and select **New Naming Rules**.



b. Specify a name, for example, **B2_Cfg0TestNamingRule**.

c. Click **Add** and specify a pattern, for example, **"CW"NNNN**.



d. Select the **Generate counters?** check box.

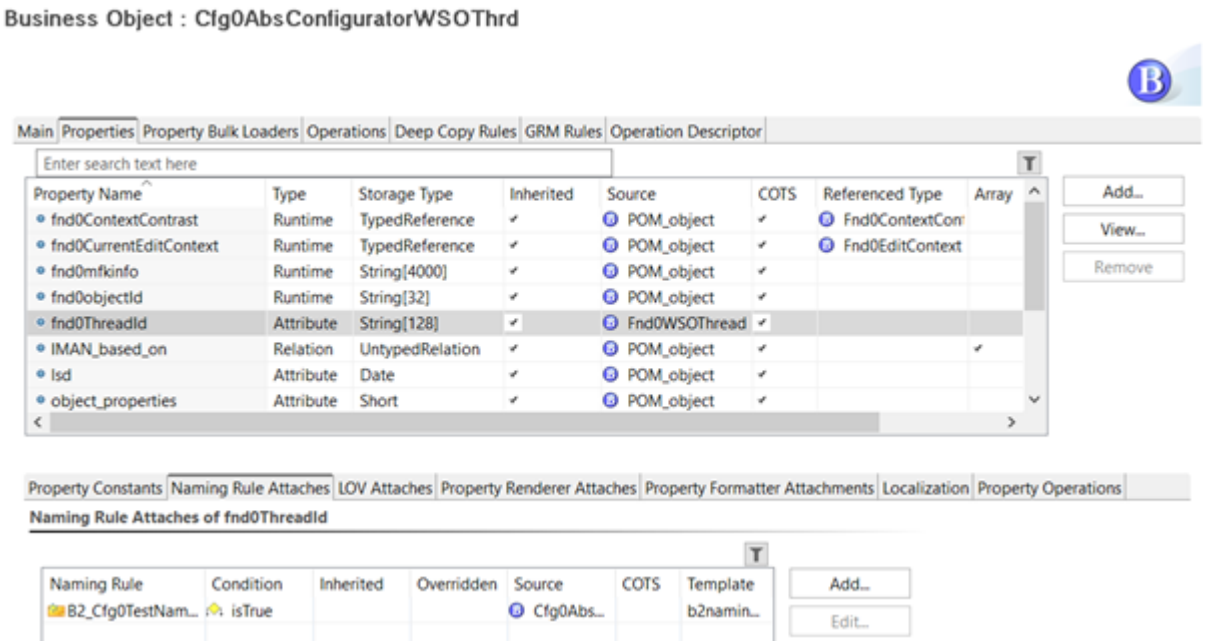
e. Specify the initial value as **CW0001** and the maximum value as **CW9999**.

f. To add the naming rule, click **Finish**.

3. Add the naming rule you created by modifying the **Cfg0ABsConfiguratorWSOThrd** business object.

   a. Choose **Find Element** on the main toolbar and type **Cfg0ABsConfiguratorWSOThrd**.

   b. Choose the **Properties** tab and select the **fnd0ThreadId** attribute.



c. To add a naming rule, choose the **Naming Rule Attaches** tab and click **Add**.

d. In the **Naming Rule** box, click **Browse**, select the naming rule you created in Step 2, and click **Finish**.

4. Generate the software package to apply the changes.

a. Click **BMIDE→Save Data Model**.

b. Click **BMIDE** > **Generate Software Package** and click **Next** in the **Recommendations for Backing up Your Template Project Source Files** dialog box.

c. In the **Generate Software Package** dialog box, note down the path in **Target folder** and click **Finish**.

Example:

*BMIDE***workspace\14000.0.0***b2namingrule**\output\wntx64\packaging\full_update***b2naming*rule*_wntx64_1.0_13.3**

5. Apply the changes using TEM.

a. Run TEM as an administrator.

b. Click **Next** until you reach the **Features** panel.

c. Click **Browse** and navigate to the location of the **Target folder** that you had noted down earlier.

d. Select the custom template under **Extensions**, for example, **a5nameid**.

e. Stop the pool manager service.

f. Click **Next** until you reach the **Confirmation** panel and then click **Start**.

g. Restart the pool manager service when the TEM feature installation process is complete.

6. Verify the changes in Active Workspace.

a. In Active Workspace, open a configurator context, and click the **Constraints** tab.

b. Choose **Grid Editor** and select **Add Constraint**.

c. Select **Inclusion Rule** and verify if the naming pattern is displayed.

Create naming rules for variant criteria objects

You can create naming rules to customize the naming of variant criteria objects. By default, no naming rules are available. However, you can create a naming rule with a proper sequence to identify different variant criteria objects. For example, you can create an inclusion rule that starts with *VC0001*. This number counter can keep incrementing till *VC9999*.

1. **Create a BMIDE template project**.
2. **Create a new naming rule**.
3. **Add the naming rule you created by modifying the Cfg0VariantCriteriaThread business object**.
4. **Generate the software package to apply the changes**.
5. **Apply the changes using TEM**.
6. **Verify the changes in Active Workspace**.

Procedure

1. Create a BMIDE template project.

a. Launch Business Modeler IDE (BMIDE).

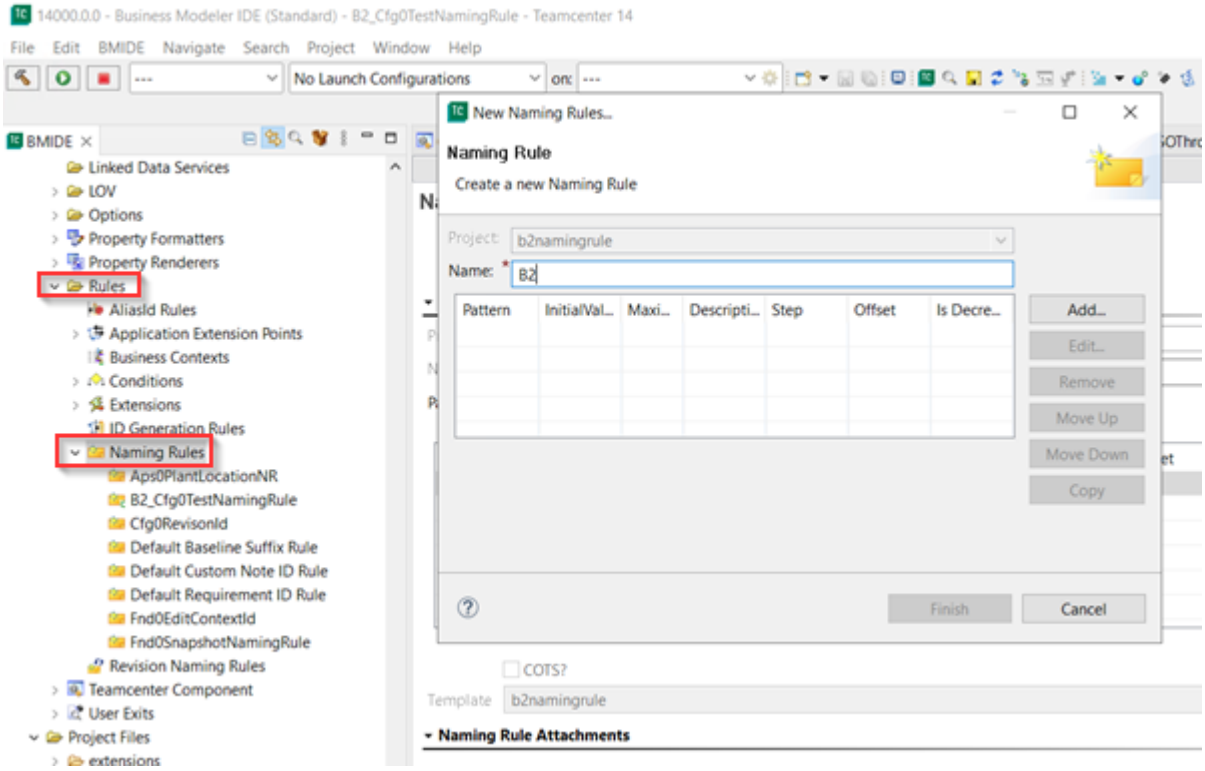
b. Create a new BMIDE template project, for example, **b2namingrulevc**.

c. In **Dependent Templates**, search for **cfg** and select **Teamcenter Configurator**.

d. Search and select all other dependent templates, and click **Finish**.

2. Create a new naming rule.

a. In Business Modeler IDE, select **Extensions\Rules\Naming Rules**, right-click, and select **New Naming Rules**.



b. Specify a name, for example, **B2_Cfg0TestNamingRuleVC**.

c. Click **Add** and specify a pattern, for example, **"VC"NNNN**.

d. Select the **Generate counters?** check box.

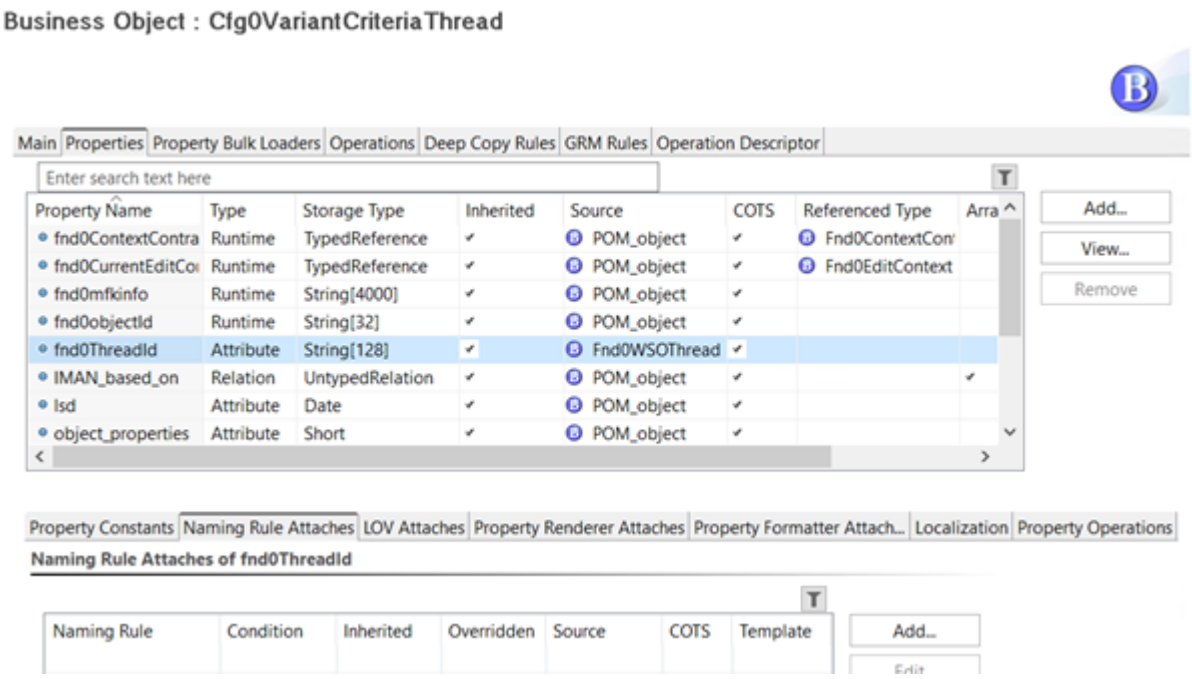
e. Specify the initial value as **VC0001** and the maximum value as **VC9999**.

f. To add the naming rule, click **Finish**.

3. Add the naming rule you created by modifying the **Cfg0VariantCriteriaThread** business object.

a. Choose **Find Element** on the main toolbar and type **Cfg0VariantCriteriaThread**.

b. Choose the **Properties** tab and select the **fnd0ThreadId** attribute.



c. To add a naming rule, choose the **Naming Rule Attaches** tab and click **Add**.

d. In the **Naming Rule** box, click **Browse**, select the naming rule you created in Step 2, and click **Finish**.

4. Generate the software package to apply the changes.

a. Click **BMIDE→Save Data Model**.

b. Click **BMIDE** > **Generate Software Package** and click **Next** in the **Recommendations for Backing up Your Template Project Source Files** dialog box.

c. In the **Generate Software Package** dialog box, note down the path in **Target folder** and click **Finish**.

Example:

*BMIDE***workspace\14000.0.0***b2namingrulevc**\output\wntx64\packaging\full_update***b2naming*rulevc**_wntx64_1.0_13.3**

5. Apply the changes using TEM.

a. Run TEM as an administrator.

b. Click **Next** until you reach the **Features** panel.

c. Click **Browse** and navigate to the location of the **Target folder** that you had noted down earlier.

d. Select the custom template under **Extensions**, for example, **a5nameid**.

e. Stop the pool manager service.

f. Click **Next** until you reach the **Confirmation** panel and then click **Start**.

g. Restart the pool manager service when the TEM feature installation process is complete.

6. Verify the changes in Active Workspace.

a. In Active Workspace, open a configurator context, and click the **Variants** tab.

b. Click **Add** and verify if the naming pattern is displayed correctly.

Customize matrix constraints

The product owner or product planner creates matrix constraints in the configurator context. The matrix is a convenient way to create constraints at the beginning of the product cycle. By providing standard, optional, default, excluded, and mandatory options, it reduces the complexity of constraints.

As an administrator, you can customize the options provided for matrix constraints in the **Matrix** tab in Active Workspace. On the user interface, standard, optional, default, excluded, and mandatory options are displayed as **S**, **O**, **D**, **E**, and **M** respectively by default. You can customize these and their descriptions by using Business Modeler IDE. You can also customize the localization of these options.

After you create the matrix and select the appropriate matrix constraint options for the groups and features, they are represented in the **Subject** column as an expression. For example, **FuelType {Diesel:S, Petrol:E}; Engine{3LitreDiesel:S, 2LitrePetrol:E}; ExteriorColor {NeptuneBlue:S}**. In this example, families **FuelType**, **Engine**, and **ExteriorColor** are separated using **;** separators. Similarly, features are followed by **:** and then the option. You can customize these separators by using BMIDE.

Customize matrix constraint options

You can customize the options provided for matrix constraints in the **Matrix** tab in Active Workspace. On the user interface, standard, optional, default, excluded, and mandatory options are displayed as **S**, **O**, **D**, **E**, and **M** respectively by default. You can customize these and their descriptions by using Business Modeler IDE. You can also customize the localization of these options.

1. **Create a BMIDE template project**.
2. **Customize the matrix constraint options by modifying the Cfg0FeatureDisposition LOV**.
3. **Generate the software package to apply the changes**.
4. **Apply the changes using TEM**.
5. **Verify the changes in Active Workspace**.

Procedure

1. Create a BMIDE template project.

   a. Launch Business Modeler IDE (BMIDE).

   b. Create a new BMIDE template project, for example, **c2matrixrules**.

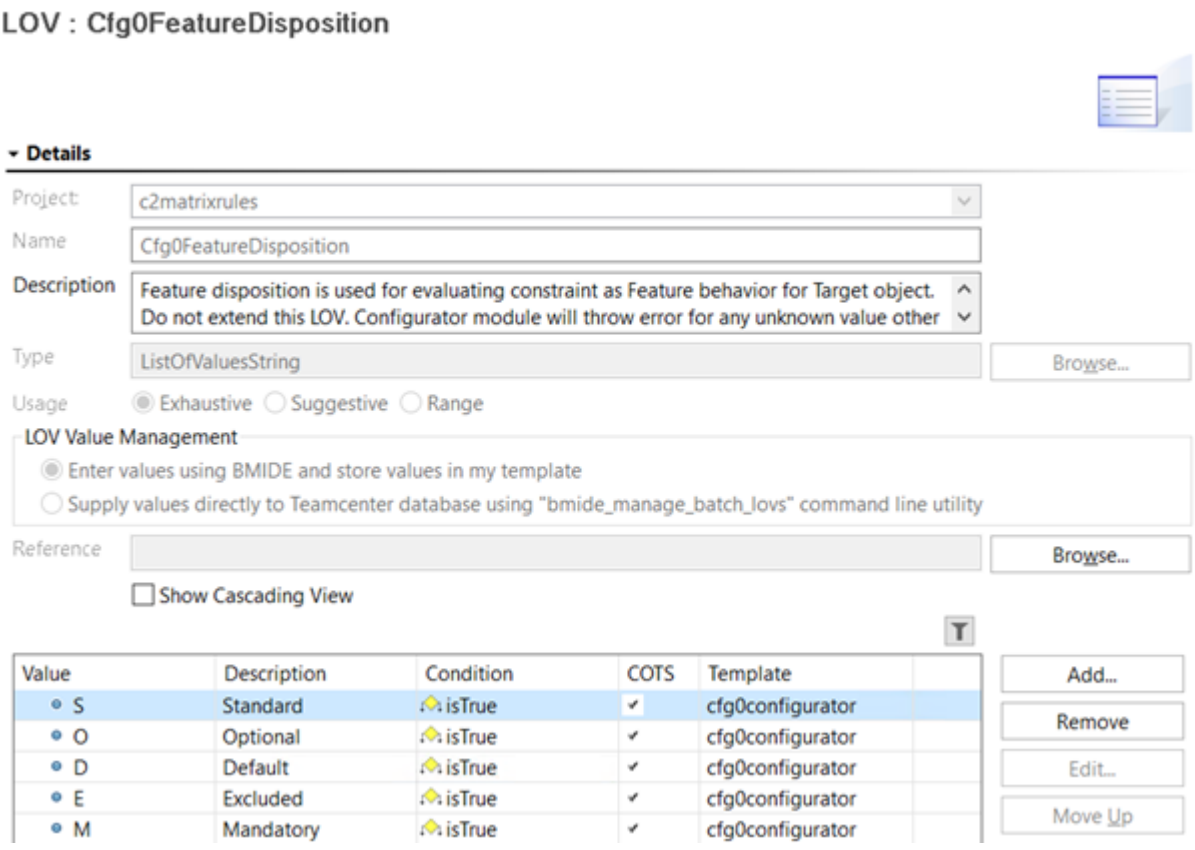
   c. In **Dependent Templates**, search for **cfg** and select **Teamcenter Configurator**.

   d. Search and select all other dependent templates and click **Finish**.

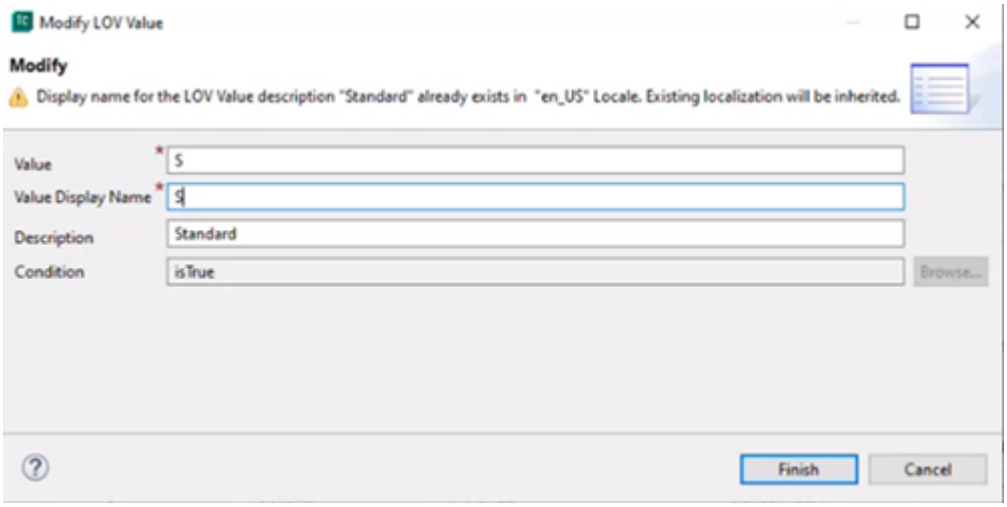
2. Customize the matrix constraint options by modifying the **Cfg0FeatureDisposition** LOV.

   a. In Business Modeler IDE (BMIDE), open the **Extensions\LOV\Classic LOV** folder.

   b. Open the **Cfg0FeatureDisposition** LOV.



   c. To edit a value, select it and click the **Edit** button and change as appropriate. You can change the value and the description.



3. Generate the software package to apply the changes.

   a. Click **BMIDE→Save Data Model**.

   b. Click **BMIDE** > **Generate Software Package** and click **Next** in the **Recommendations for Backing up Your Template Project Source Files** dialog box.

   c. In the **Generate Software Package** dialog box, note down the path in **Target folder** and click **Finish**.

      Example:

      *BMIDE***workspace****14000.0.0***c2matrixrules***output****wntx64****packaging****full_update***c2matrixrules*_**wntx64_1.0_13.3**

4. Apply the changes using TEM.

   a. Run TEM as an administrator.

   b. Click **Next** until you reach the **Features** panel.

   c. Click **Browse** and navigate to the location of the **Target folder** that you had noted down earlier.

   d. Select the custom template under **Extensions**, for example, **a5nameid**.

   e. Stop the pool manager service.

   f. Click **Next** until you reach the **Confirmation** panel and then click **Start**.

   g. Restart the pool manager service when the TEM feature installation process is complete.

5. Verify the changes in Active Workspace.

    a. In Active Workspace, open a configurator context, and click the **Constraints** tab.

    b. Choose the **Matrix** in the secondary view and select **Add Constraint**.

    c. Specify the message and click **Add**.

    d. Add the variability content for the condition and the subject as appropriate.

    e. Click **Edit** to make selections.

    f. In **Subject**, verify if the options you customized are displayed correctly.

Customize separators in the matrix constraint expression

You can customize the separators in the **Subject** box of the matrix constraint.

After you create the matrix and select the appropriate matrix constraint options for the groups and features, they are represented in the **Subject** box as an expression. For example, **FuelType {Diesel:S, Petrol:E}; Engine{3LitreDiesel:S, 2LitrePetrol:E}; ExteriorColor {NeptuneBlue:S}**. In this example, families **FuelType**, **Engine**, and **ExteriorColor** are separated using **;** separators. Similarly, features are followed by **:** and then the option. You can customize these separators by using Business Modeler IDE.

1. **Create a BMIDE template project**.
2. **Customize separators in the matrix constraint expression by modifying the Cfg0MatrixRuleSubjectSeparators LOV.**.
3. **Generate the software package to apply the changes**.
4. **Apply the changes using TEM**.
5. **Verify the changes in Active Workspace**.

Procedure

1. Create a BMIDE template project.

    a. Launch Business Modeler IDE (BMIDE).

    b. Create a new BMIDE template project, for example, **c2matrixrules**.

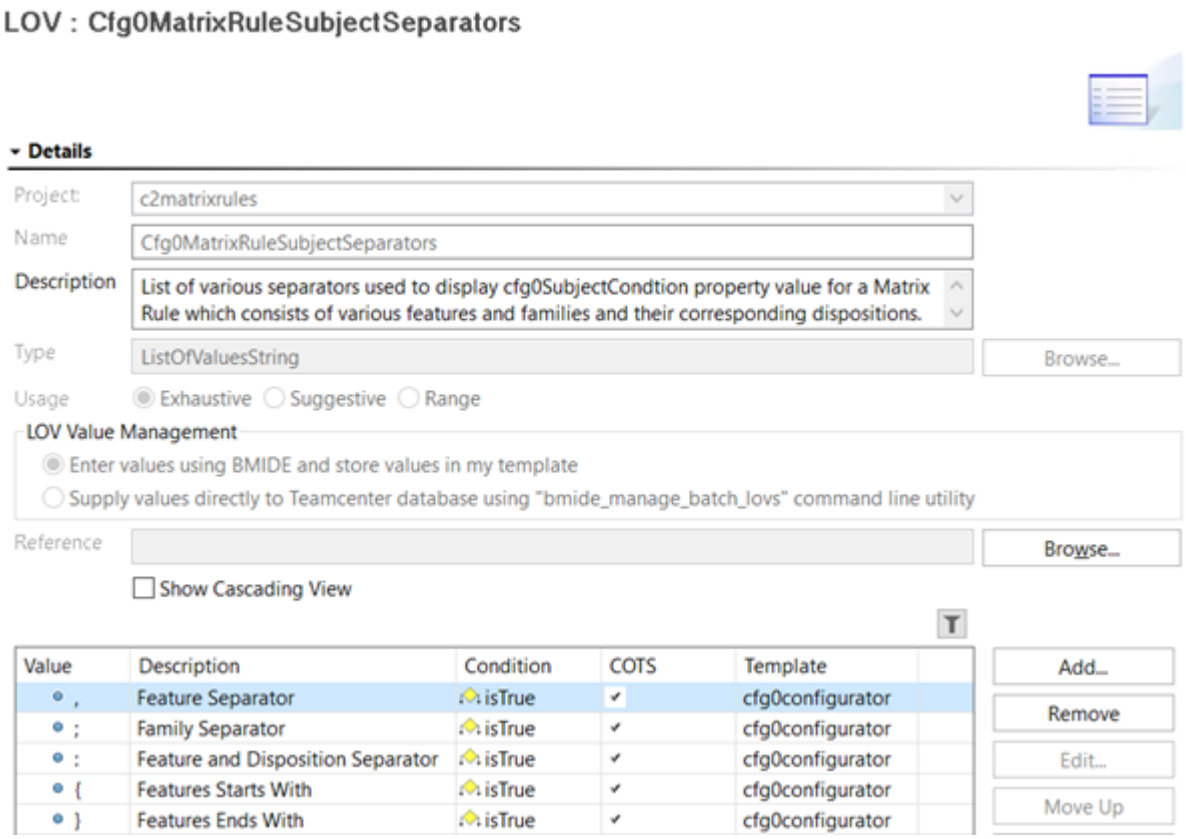
    c. In **Dependent Templates**, search for **cfg** and select **Teamcenter Configurator**.

    d. Search and select all other dependent templates and click **Finish**.

2. Customize separators in the matrix constraint expression by modifying the **Cfg0MatrixRuleSubjectSeparators** LOV.

a. In Business Modeler IDE (BMIDE), open the **Extensions\LOV\Classic LOV** folder.

b. Open the **Cfg0MatrixRuleSubjectSeparators** LOV.



c. To edit a value, select it, click the **Edit** button, and use a different separator as appropriate.

3. Generate the software package to apply the changes.

a. Click **BMIDE→Save Data Model**.

b. Click **BMIDE** > **Generate Software Package** and click **Next** in the **Recommendations for Backing up Your Template Project Source Files** dialog box.

c. In the **Generate Software Package** dialog box, note down the path in **Target folder** and click **Finish**.

Example:

*BMIDE***workspace\14000.0.0***c2matrixrules**\output\wntx64\packaging\full_update***c2matrixrules**_**wntx64_1.0_13.3**

4. Apply the changes using TEM.

a. Run TEM as an administrator.

b. Click **Next** until you reach the **Features** panel.

c. Click **Browse** and navigate to the location of the **Target folder** that you had noted down earlier.

d. Select the custom template under **Extensions**, for example, **a5nameid**.

e. Stop the pool manager service.

f. Click **Next** until you reach the **Confirmation** panel and then click **Start**.

g. Restart the pool manager service when the TEM feature installation process is complete.

5. Verify the changes in Active Workspace.

a. In Active Workspace, open a configurator context, and click the **Constraints** tab.

b. Choose the **Matrix** in the secondary view and select **Add Constraint**.

c. Specify the message and click **Add**.

d. Add the variability content for the condition and the subject as appropriate.

e. Click **Edit** to make selections.

f. Select the appropriate conditions and choose the appropriate options for the subject.

g. Click **Save Edits**.

h. View the **Subject** column of the matrix constraint you created and verify if the separators you customized are displayed correctly.

Customize formula authoring in Teamcenter configurator

Customize formula authoring

Internal formulas are essential constructs within the Teamcenter configurator framework. They define relationships and behaviors between different elements of the configurator context, such as models and features. Internal formulas use feature or family IDs along with predefined tokens for operators, such as **&** for AND, **!** for NOT, and **|** for OR, to specify constraints, conditions, and rules.

In contrast, *display formulas* represent expressions and conditions in a human-readable format. Unlike internal formulas that may use feature IDs, display formulas are more comprehensible as they use family

or feature names and customizable tokens for operators as specified by the user. They are typically shown in the constraints table.

To illustrate configurator variability within a configurator context, consider the following example. The system has a hierarchical structure of features, where each feature or family has a unique identifier (ID).

Variability	
Name	**ID**
Model	**Model_ID**
-LXI	**LXI_ID**
-VXI	**VXI_ID**
-ZXI	**ZXI_ID**
Engine	**Engine_ID**
-Petrol	**Petrol_ID**
-Diesel	**Diesel_ID**
-CNG	**CNG_ID**
Speaker	**Speaker_ID**
-Bose	**Bose_ID**
-Sony	**Sony_ID**
-JBL	**JBL_ID**

An include rule constraint, such as **Incl_Rule_001** can be authored in various ways within the Teamcenter configurator framework. This rule defines how specific features or conditions should be included within the configurator context. Here are some common methods to author an include rule constraint:

- *Grid Editor*

  It offers a straightforward and user-friendly interface for configuring constraints. In this approach, you define the subject and conditions for a constraint by specifying the feature selections in a tabular grid format. This method is intuitive, especially for users who prefer a visual representation of conditions and subjects.

- *Formula suggester*

  It allows for more complex expressions by enabling the use of conditional logic and feature names within formulas. The formula is authored using the display formula. For example, to create a rule that states, `If the Model is LXI, then the Engine must be Petrol and the Speaker must be Bose`, you can author it as follows:

  Condition: `Model=LXI`

Subject: `Engine=Petrol AND Speaker=Bose`

To use the **AND** operator in the formula, you must configure the text server value of **k_variant_op_and** to **AND**. This operator and other customizable tokens allow for flexible and tailored formula authoring based on specific needs.

The following is a list of tokens that can be customized and used in formula authoring:

- **k_variant_op_and**: Represents logical AND operation
- **k_variant_op_or**: Represents logical OR operation
- **k_variant_op_not**: Represents logical NOT operation
- **k_variant_l_bracket**: Represents left bracket (for grouping)
- **k_variant_r_bracket**: Represents right bracket (for grouping)
- **k_variant_op_is_equal**: Represents equality operation
- **k_variant_op_not_equal**: Represents inequality operation
- **k_variant_op_gt**: Represents greater than operation
- **k_variant_op_lt**: Represents less than operation
- **k_variant_op_gt_eq**: Represents greater than or equal to operation
- **k_variant_op_lt_eq**: Represents less than or equal to operation

Customize the formula handling mechanisms for set expression

User exits in the Teamcenter Product Configurator allow for customizing the formula handling mechanisms. These exits operate on the **Cfg0ConfiguratorPerspective** business object and can be used to define custom behaviors for converting and processing formulas. To access these operations, open the **Cfg0ConfiguratorPerspective** business object and navigate to the **Operations** tab, where you will find operations such as:

BMF_USER_CFG0_convert_teamcenter_formula_to_custom_formula

This operation uses the following API as defined in the **cfg0configurator_user_exits** file:

```
extern CFG0CONFIGURATOR_API int
USER_CFG0_convert_teamcenter_formula_to_custom_formula(
  tag_t cfg0_perspective, /**< (I) The Cfg0ConfiguratorPerspective
object for which
      Teamcenter formula to custom formula conversion is expected. */
  const char* tc_formula, /**< (I) Teamcenter formula string */
  char** custom_formula /**< (OF) Converted custom formula */ );
```

The purpose of this API is to convert a Teamcenter formula into a custom formula format. When this user exit is defined or overridden, it affects all read operations that retrieve variant formulas across configurator rules, design elements, and more. The converted custom formula will be displayed in views such as the **Variant Expression Editor** and **Configurator Rule** views in the rich client. This API does not affect variant expression authoring as long as the input formulas follow the Teamcenter formula format.

The purpose of this API is to convert a Teamcenter formula into a custom formula format. When this user exit is defined or overridden, it affects all read operations that retrieve variant formulas across configurator rules, design elements, and more. The converted custom formula will be displayed in views such as the **Variant Expression Editor** and **Configurator Rule** views in the rich client. This API does not affect variant expression authoring as long as the input formulas follow the Teamcenter formula format.

The parameters for the API are as follows:

- **cfg0_perspective** (tag): Specifies the perspective for the configurator context, including revision rules and product contexts to manage variability.
- **tc_formula** (string): The Teamcenter formula that needs to be converted into the custom formula format.
- **custom_formula** (string): The resulting custom formula for the input Teamcenter formula.

BMF_USER_CFG0_convert_custom_formula_to_teamcenter_formula

This operation uses the following API:

```
extern CFG0CONFIGURATOR_API int
USER_CFG0_convert_custom_formula_to_teamcenter_formula(
  tag_t cfg0_perspective, /**< (I) The Cfg0ConfiguratorPerspective
object for which
      Custom formula to Teamcenter formula conversion is expected. */
  const char* custom_formula, /**< (I) Custom formula string */
  char** tc_formula /**< (OF) Converted Teamcenter formula */ );
```

This API converts custom formulas into the Teamcenter format. When overridden, it impacts authoring operations for variant expressions, including saving variant rules and configurator rules. The converted formula will be used in various views within the Product Configurator rich client. Display functionality remains unaffected.

The parameters used by this API are as follows:

- **cfg0_perspective** (tag): Specifies the perspective for the configurator context, managing configuration information such as revision rules and contexts.
- **custom_formula** (string): Specifies the custom formula to be converted to the Teamcenter formula format.
- **tc_formula** (string): Specifies the resulting Teamcenter formula for the input custom formula.

A return value of **0** indicates successful formula conversion. Otherwise, an error code is returned.

Define expressions using setter APIs

The Teamcenter configurator allows setting expressions for constraints using setter APIs. These APIs typically require the *internal formula* for authoring.

Example:

```
AOM_set_value_string(includeRuleObjectTag, "cfg0SubjectCondition",
    "Engine_ID=Petrol_ID & Speaker_ID=Bose_ID");
```

In this example, an internal formula is used, meaning that feature or family IDs must be specified. Special character tokens are used for logical operations, such as **&** for AND, **|** for OR, and **!** for NOT. Recent enhancements allow for more flexible formula authoring using custom tokens. For instance, you can replace **&** with **+** if the value of **k_variant_op_and** has been customized to **+**. To enable this capability, set the **Cfg0EnableCustomFormulaAuthoring** site preference to **true**.

The formula can be authored as follows:

```
AOM_set_value_string(includeRuleObjectTag, "cfg0SubjectCondition",
    "Engine_ID=Petrol_ID + Speaker_ID=Bose_ID");
```

> Note:
>
> Formulas cannot use multi-character or multi-byte tokens for setting expressions using setter APIs, for example, **AND**. Only single-character tokens are allowed, for example, **+**, **-**, **|**.
>
> Certain characters, such as **'**and **** hold special meanings and cannot be used as token values.

8. Create or modify column configuration in Active Workspace

Modify column attributes for the Table view in the Constraints tab

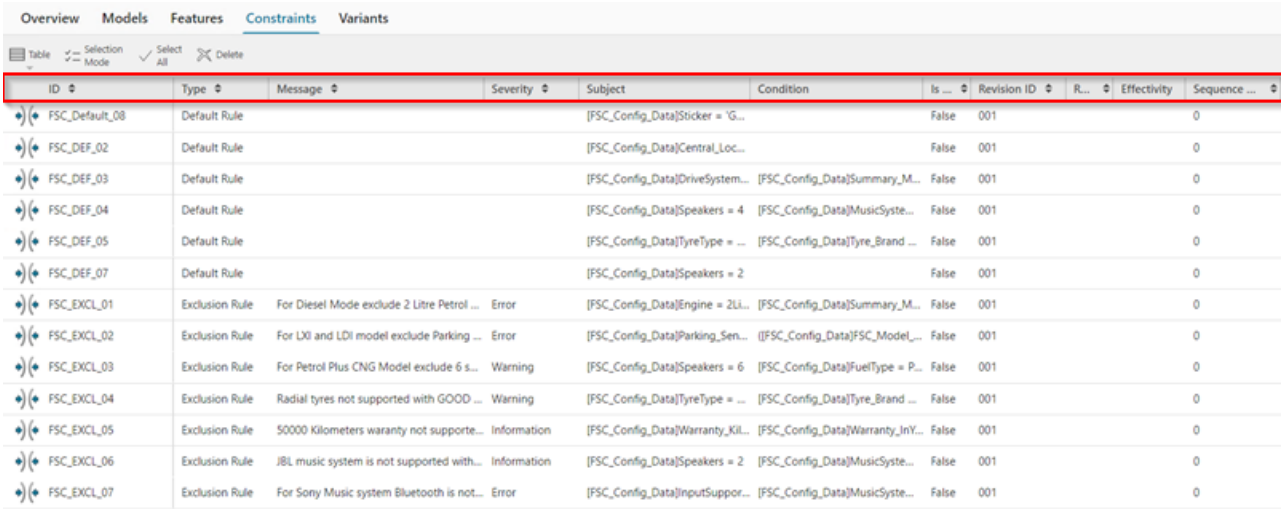
As an administrator, you can modify column attributes for the **Table** view in the **Constraints** tab by modifying the **Pca0ConstraintsUiConfigCots.xml** column configuration (COTS) file.

For example, the **Constraint Set** column is not shown by default in the **Constraints** view. The following procedures show how to add this column to the view.

A configurator administrator or a designated user creates constraint sets to group multiple constraints in a set.

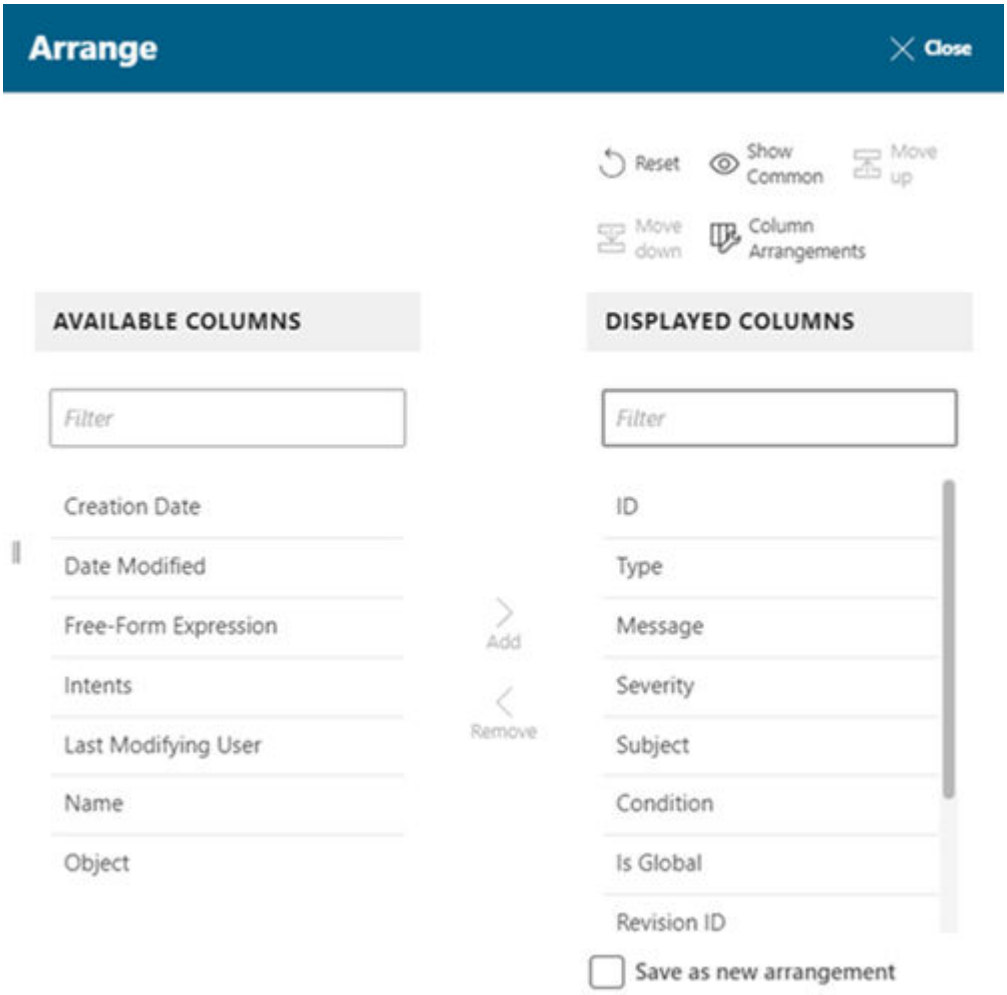
Procedure

1. Open a configurator context or configurator dictionary in Active Workspace, choose the **Constraints** tab, and select the **Table** view. It displays 11 columns by default.



2. Click **Table Settings** > **Arrange**.

   The **AVAILABLE COLUMNS** section displays seven additional columns that can be displayed in the **Table** view of the **Constraints** tab.



3. Open the *TC_ROOT***install\pca0awconfigurator\data\Pca0ConstraintsUiConfigCots.xml** column configuration file.

   The number of columns available in the **Table** view of the **Constraints** tab is determined by this file.

   Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Import>
    <Client abbreviation="AWClient" name="AWClient">
        <ClientScope hostingClientName="" name="Pca0Constraints"
uri="Pca0Constraints">
            <ColumnConfig columnConfigId="Pca0ConstraintsColConfig"
sortBy="-1" sortDirection="Ascending">
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0ObjectId" width="250"/>
                <ColumnDef objectType="Cfg0AbsRule"
```

```
propertyName="object_type" width="100"/>
                <ColumnDef objectType="Cfg0AbsConstraintRule"
propertyName="cfg0Message" width="300"/>
                <ColumnDef objectType="Cfg0AbsConstraintRule"
propertyName="cfg0Severity" width="100"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0SubjectCondition" width="300"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0ApplicabilityCondition" width="300"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0IsGlobal" width="50"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="fnd0RevisionId" width="100"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="release_status_list" width="50"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0Effectivity" width="200"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0Sequence" width="100"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="creation_date" width="100" hidden="true"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="last_mod_date" width="100" hidden="true"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="last_mod_user" width="150" hidden="true"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="object_string" width="300" hidden="true"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0ExpScript" width="300" hidden="true"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0Intents" width="100" hidden="true"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="object_name" width="250" hidden="true"/>
            </ColumnConfig>
        </ClientScope>
    </Client>
</Import>
```

In the above example, this column configuration file contains 11 visible columns and seven hidden columns.

When you set a property to **hidden=true**, that property is available to the user if they want to add it, but it is not displayed by default. While a property remains hidden, it is not retrieved from the database, which improves table rendering time.

4. To modify the column configurations by adding additional properties, copy the **Pca0ConstraintsUiConfigCots.xml** file to the **C:\Temp** directory as reference to create the column configuration.

To modify an existing column configuration or to create a new definition, you must define it in an XML file and then import it. Creating the XML definition is easiest if you export the existing column configurations for reference, copy one, and then modify it for import.

For more information, see *Create or modify a column configuration* in *Active Workspace Customization*.

5. Open the **C:\Temp\Pca0ConstraintsUiConfigCots.xml** and specify the additional property you want to be displayed in the GUI.

Property name in the COTS file	Property name in the GUI
tc_xrt_ConstraintSet	Constraint Set

Example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Import>
    <Client abbreviation="AWClient" name="AWClient">
        <ClientScope hostingClientName="" name="Pca0Constraints"
uri="Pca0Constraints">
            <ColumnConfig columnConfigId="Pca0ConstraintsColConfig"
sortBy="-1" sortDirection="Ascending">
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0ObjectId" width="250"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="object_type" width="100"/>
                <ColumnDef objectType="Cfg0AbsConstraintRule"
propertyName="cfg0Message" width="300"/>
                <ColumnDef objectType="Cfg0AbsConstraintRule"
propertyName="cfg0Severity" width="100"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0SubjectCondition" width="300"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0ApplicabilityCondition" width="300"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0IsGlobal" width="50"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="fnd0RevisionId" width="100"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="release_status_list" width="50"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0Effectivity" width="200"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0Sequence" width="100"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="creation_date" width="100" hidden="true"/>
                <ColumnDef objectType="Cfg0AbsRule"
```

```
propertyName="last_mod_date" width="100" hidden="true"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="last_mod_user" width="150" hidden="true"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="object_string" width="300" hidden="true"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0ExpScript" width="300" hidden="true"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0Intents" width="100" hidden="true"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="object_name" width="250" hidden="true"/>

                <!-- ADD NEW PROPERTY AS FOLLOWS -->

                <ColumnDef objectType="Cfg0AbsRule" filterable="true"
propertyName="REFBY(contents,
                        Cfg0ConstraintSet).object_name"
columnName="tc_xrt_ConstraintSet" width="150"/>
            </ColumnConfig>
        </ClientScope>
    </Client>
</Import>
```

In the above example, specify the type of object for which this column is valid and the name of the property whose value will be displayed.

For more information, see *Syntax for column configuration* in *Active Workspace Customization*.

6. Use the **import_uiconfig** utility to import your new XML file.

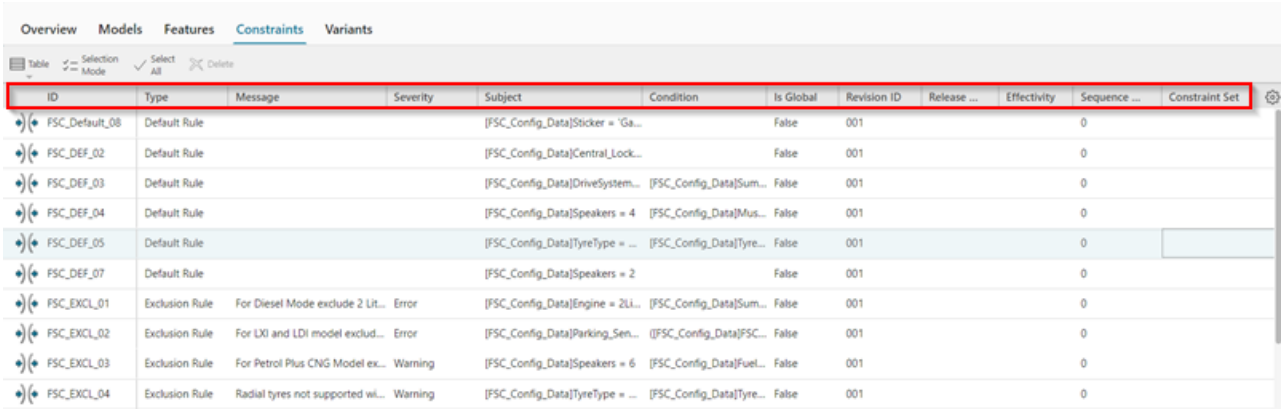
   For more information, see **import_uiconfig** in *Teamcenter Utilities*.

   Example to import configuration specified in the XML file for multiple roles, **Designer** and **engineer**:

   import_uiconfig -u=*User_ID* **-p=***password* **-g=***group*
   -file=*C:\Temp\Pca0ConstraintsUiConfigCots.xml* **-for_role=***Designer,engineer*

7. If you have Active Workspace open, log off and log on again.

   Open a configurator context or configurator dictionary in Active Workspace, choose the **Constraints** tab, and select the **Table** view. It now displays 12 columns. 11 default columns and the column you added in **step 4**.



Modify the header properties in the Grid Editor of the Constraints tab

As an administrator, you can modify column attributes for the header properties in the **Constraints** tab by modifying the **Pca0ConstraintsHeaderPropertiesConfigCots.xml** column configuration (COTS) file.

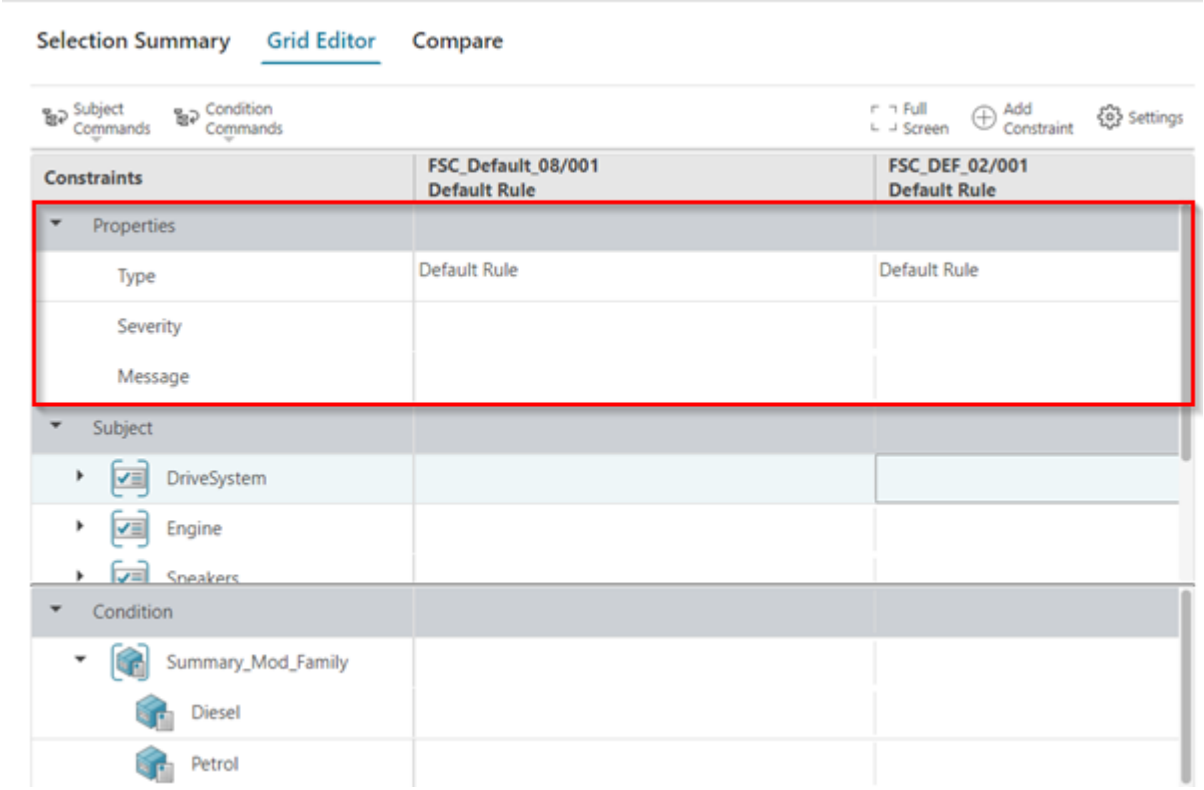
Procedure

1. Open a configurator context or configurator dictionary in Active Workspace, choose the **Constraints** tab, and select a few constraints.

2. Select **Table with Summary** view and choose **Grid Editor**.

3. Choose **Settings** and enable the **Show Constraints Information Content in Grid** toggle.

   When you enable this toggle, the **Grid Editor** displays three properties by default. They are **Type**, **Severity**, and **Message**.



4. Open the *TC_ROOT***install\pca0awconfigurator\data\Pca0ConstraintsHeaderPropertiesConfigCots.xml** file.

   This file determines the properties that are available by default.

Property name in the COTS file	Property name in the GUI
object_type	**Type**
cfg0Severity	**Severity**
cfg0Message	**Message**

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Import>
    <Client abbreviation="AWClient" name="AWClient">
        <ClientScope
hostingClientName="" name="Pca0ConstraintsHeaderProperties"
uri="Pca0ConstraintsHeaderProperties">
            <ColumnConfig
columnConfigId="Pca0ConstraintsHeaderPropConfig" sortBy="-1"
sortDirection="Ascending">
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="object_type" width="10"/>
                <ColumnDef objectType="Cfg0AbsConstraintRule"
```

```
propertyName="cfg0Severity" width="10"/>
                <ColumnDef objectType="Cfg0AbsConstraintRule"
propertyName="cfg0Message" width="10"/>
            </ColumnConfig>
        </ClientScope>
    </Client>
</Import>
```

5. To add more properties in the GUI, you can modify the COTS file. To do so, copy the *TC_ROOT***install\pca0awconfigurator\data\Pca0ConstraintsHeaderPropertiesConfigCots.xml** file to the **C:\Temp** directory.

6. Add two attributes as follows in the **C:\Temp\Pca0ConstraintsHeaderPropertiesConfigCots.xml** file to add two new properties in the GUI.

Property name in the COTS file	Property name in the GUI
fnd0RevisionId	**Rev ID**
cfg0ObjectId	**ID**

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Import>
    <Client abbreviation="AWClient" name="AWClient">
        <ClientScope
hostingClientName="" name="Pca0ConstraintsHeaderProperties"
uri="Pca0ConstraintsHeaderProperties">
            <ColumnConfig
columnConfigId="Pca0ConstraintsHeaderPropConfig" sortBy="-1"
sortDirection="Ascending">
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="object_type" width="10"/>
                <ColumnDef objectType="Cfg0AbsConstraintRule"
propertyName="cfg0Severity" width="10"/>
                <ColumnDef objectType="Cfg0AbsConstraintRule"
propertyName="cfg0Message" width="10"/>

                <!-- ADD TWO NEW PROPERTIES AS FOLLOWS -->

                <ColumnDef objectType="Cfg0AbsRule"
propertyName="fnd0RevisionId" width="10"/>
                <ColumnDef objectType="Cfg0AbsRule"
propertyName="cfg0ObjectId" width="10"/>
                </ColumnConfig>
        </ClientScope>
    </Client>
</Import>
```

7. Use the **import_uiconfig** utility to import your new XML file.

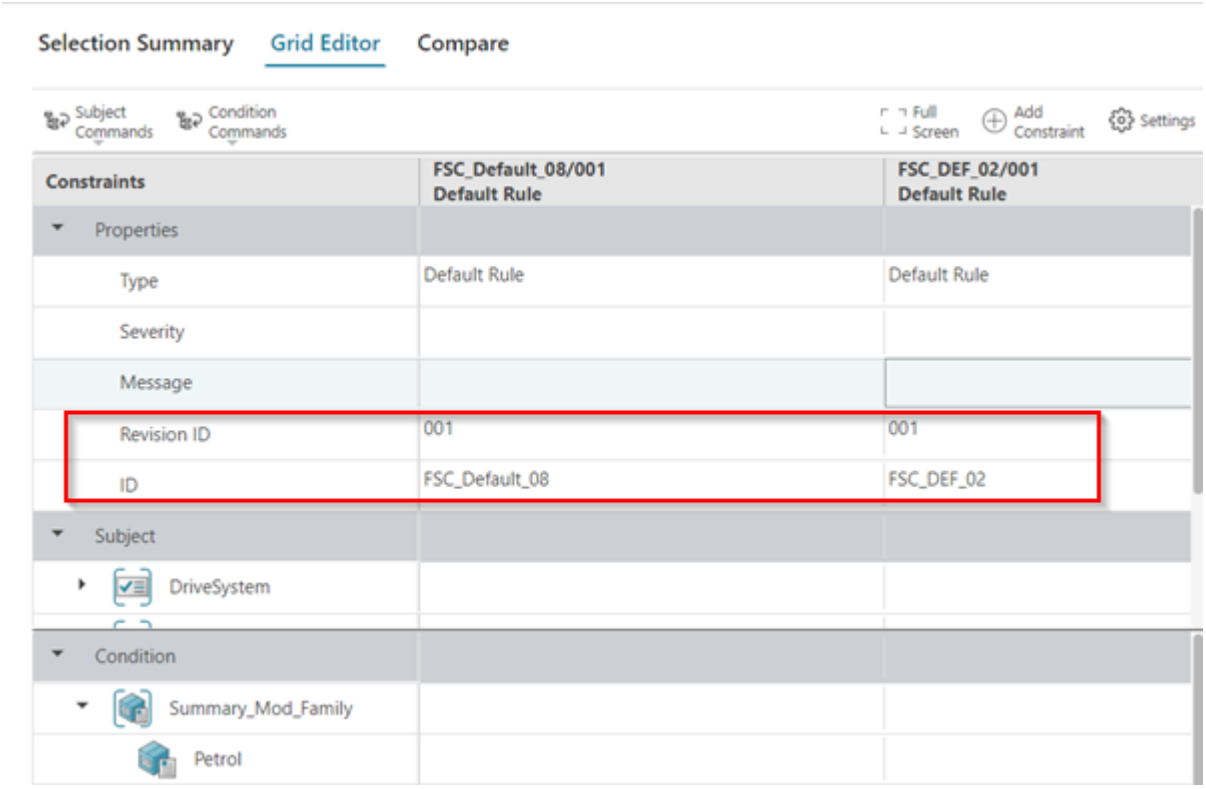
For more information, see **import_uiconfig** in *Teamcenter Utilities*.

Example to import configuration specified in the XML file for multiple roles, **Designer** and **engineer**:

import_uiconfig -u=*User_ID* **-p=***password* **-g=***group*
-file=*C:\Temp\Pca0ConstraintsHeaderPropertiesConfigCots.xml* **-for_role=***Designer,engineer*

8. If you have Active Workspace open, log off and log on again.

   Open the **Grid Editor**. It displays five properties, that is, two properties in addition to the default three properties. They are **Type**, **Severity**, **Message**, **Revison ID**, and **ID**. The first three are default properties.



Modify columns attributes for views in the Models, Features, or Variants tab

As an administrator, you can modify the column attributes by editing the following column configuration files:

Views, tabs, and panels	Location of COTS file
Tree view in the **Models** tab	**Pca0VariabilityExplorerProducts** client scope in *TC_ROOT***install\pca0awconfigurator\data\Pca0VariabilityExplorerUiConfigCots.xml**
Tree view in the **Features** tab	**Pca0VariabilityExplorerFeatures** client scope in *TC_ROOT***install\pca0awconfigurator\data\Pca0VariabilityExplorerUiConfigCots.xml**
Table view in the **Variants** tab	*TC_ROOT***install\pca0awconfigurator\data\Pca0VariantRuleUiConfigCots.xml**
Constraints tab > **Grid Editor** tab in secondary view > **Add Families and Features** (command) > **Add Families and Features** panel	**Pca0VariabilityInConstraints** client scope in *TC_ROOT***install\pca0awconfigurator\data\Pca0VariabilityExplorerUiConfigCots.xml**

For more information, see What is column configuration? in *Active Workspace Customization*.

9. Using Teamcenter workflows in Product Configurator

Create workflows to release configurator data

Administrators use specific workflow handlers to ensure that when you release configurator features, you can also automatically include their families and configurator allocations in the same workflow process.

You attach the Work in Progress (WIP) or Latest Released revisions of configurator data to the workflow process. This allows you to configure the handlers to:

- Attach the **Latest Released** revisions
- Attach them as reference attachments, if specified in a handler.

For example, you can configure a workflow to initially look for WIP revisions to attach them as targets, and then look again for released revisions to attach them as a reference. As the result, you see the following:

- If the latest released revision of a feature is attached as a reference, then Teamcenter finds the WIP allocations.
- If a WIP configurator rule references released features, then Teamcenter finds the WIP allocations.

Attach the following	Use the workflow handler
Configurator rules that reference a feature or family	**CFG0-attach-constraint-rules**
Families and features that are referenced by a configurator rule	**CFG0-attach-rule-variability**
Families that are referenced by features	**CFG0-attach-families**
Groups that reference families	**CFG0-attach-familygroups**
Configurator allocation objects that reference features, families, or groups	**CFG0-attach-allocations**
Creates the report of constraint conflicts for a given variant rule and its subtypes.	**CFG0-find-constraint-conflict**

Workflow process example for a configurator rule in a configurator context

Create a new workflow based on the Cfg0 Workflow template (administrator task)

1. Open Workflow Designer on Rich Client and create a new **Cfg0 Workflow** workflow process template.

2. Add the following action handlers to the **Start** and **Complete** actions of the **Do** task. Ensure the handler argument specifies a status of **TCM Released**.

   TCM Released is one of the standard status values provided by default with Teamcenter.

   The following displays both the default and configured arguments.

Action	Workflow Handler	Argument	Value
Start	**EPM-create-status**	-status	TCM Released
	CFG0-attach-rule-variability	-attachment	target (default value)
		-configuration	Latest Working (default)
		-attachConfiguratorContext	true (default value)
	CFG0-attach-families	-attachment	target (default value)
		-configuration	Latest Working (default)
	CFG0-attach-familygroups	-attachment	target (default value)
		-configuration	Latest Working (default)
	CFG0-attach-allocations	-attachment	target (default value)
		-configuration	Latest Working (default)
		-attachedConfiguratorContext	true
Complete	**EPM-set-status**	-action	append

3. Make the new workflow available to the users.

Run a new Cfg0 Workflow process on a configurator rule (user task)

In this scenario, the administrator configured the **Cfg0 Workflow** process and made it available to the users.

You create data for a configurator context **P1** with the following values.

Configurator Object Type	ID/Rev
Model Family	**Models/001**
Model	**M1/001**
Group	**Powertrain/001**
Family	**ENG/001**
Feature	**LPG/001**

You create the following availability rule **R1**:

- **LPG** is available for **M1**.

1. Select the availability rule **R1** and choose **File→New→Workflow Process**.

   The following displays the state of the workflow process and its current attachments after each step.

   - In the initial state, such as the state with which you initiate the workflow process, the only attachment is an availability rule **R1/001**. This rule is attached as a target.
   - Next, **P1** is attached because the **CFG0-attach-rule-variability** handler was run with an argument **-attachConfiguratorContext=true**.

     The **CFG0-attach-rule-variability** workflow handler attaches **Feature LPG/001** and **Model M1/001** as additional targets. Because of the **-attachConfiguratorContext=true** argument, the handler also attaches the configurator context items to which the availability rule applies as reference attachments.

     > Note:
     >
     > If you choose to run this step with the handler set to **-attachConfiguratorContext=false**, then **P1** is not attached.

   - The **CFG0-attach-allocations** workflow handler attaches allocation WIP revisions for the configurator items attached to this workflow.

Workflow step	Description	Attachments (Target)	Attachments (Reference)
1	Initiation step	Availability rule **R1**	
2	**CFG0-attach-rule-variability**	Availability rule **R1/001**	**P1**
		Feature **LPG/001**	
		Model **M1/001**	

Workflow step	Description	Attachments (Target)	Attachments (Reference)
3	**CFG0-attach-families**	Availability rule **R1/001**	**P1**
		Feature **LPG/001**	
		Model **M1/001**	
		Family **ENG/001**	
		Model family **Models/001**	
4	**CFG0-attach-familygroups**	Availability rule **R1/001**	**P1**
		Feature **LPG/001**	
		Model **M1/001**	
		Family **ENG/001**	
		Model family **Models/001**	
		Group **Powertrain/001**	
5	**CFG0-attach-allocations**	Availability rule **R1/001**	**P1**
		Feature **LPG/001**	
		Model **M1/001**	
		Family **ENG/001**	
		Model family **Models/001**	
		Group **Powertrain/001**	
		Feature allocation **LPG/001 → P1**	
		Family allocation **ENG/001 → P1**	
		Powertrain allocation **Powertrain/001 → P1**	

Workflow process example for a configurator feature in a configurator context

Create a new workflow based on the Cfg0 Workflow template (administrator task)

1. Open Workflow Designer on Rich Client and create a new **Cfg0 Workflow** workflow process template.

2. Add the following action handlers to the **Start** and **Complete** actions of the **Do** task. Ensure the handler argument specifies a status of **TCM Released**.

TCM Released is one of the standard status values provided by default with Teamcenter.

The following displays both the default and configured arguments.

Action	Workflow Handler	Argument	Value
Start	**EPM-create-status**	-status	TCM Released
	CFG0-attach-constraint-rules	-attachment	target (default value)
		-configuration	Latest Working (default)
		-attachedConfiguratorContext	false
	CFG0-attach-rule-variability	-attachment	target (default value)
		-configuration	Latest Working (default)
		-attachConfiguratorContext	true (default value)
	CFG0-attach-families	-attachment	target (default value)
		-configuration	Latest Working (default)
	CFG0-attach-familygroups	-attachment	target (default value)
		-configuration	Latest Working (default)
	CFG0-attach-allocations	-attachment	target (default value)
		-configuration	Latest Working (default)
		-attachedConfiguratorContext	true
Complete	**EPM-set-status**	-action	append

3. Make the new workflow available to the users.

Run a new Cfg0 Workflow process on a configurator value (user task)

In this scenario, the administrator configured the **Cfg0 Workflow** process and made it available to the users.

You create data for a configurator context **P1** with the following values.

Configurator Object Type	ID/Rev
Model Family	**Models/001**
Model	**M1/001**
Group	**Powertrain/001**
Family	**ENG/001**
Feature	**LPG/001**

You create the following availability rule **R1**:

- **LPG** is available for **M1**.

1. Select the feature **LPG/001** and choose **File→New→Workflow Process**.

   The following displays the state of the workflow process and its current attachments after each step.

   - The workflow handler **CFG0-attach-constraint-rules** is run with an argument **-attachedConfiguratorContext=false**. It attaches WIP revisions for all constraint rules that reference the variability in this workflow as additional targets. In this scenario, it attaches the availability rule **R1**, such as the configurator rules to attach are not filtered by the configurator context items attached to this workflow.

   - The **CFG0-attach-rule-variability** workflow handler attaches the value **LPG/001** and the model **M1/001** as additional targets. Because of its argument **-attachConfiguratorContext=true** it also attaches the configurator context items to which the availability rule applies as reference attachments.

   - The **CFG0-attach-allocations** workflow handler attaches allocation WIP revisions for the configurator context items attached to this workflow.

Workflow step	Description	Attachments (Target)	Attachments (Reference)
1	Initiation step	Feature **LPG/001**	
2	**CFG0-attach-constraint-rules**	Feature **LPG/001**	
		Availability rule **R1/001**	
3	**CFG0-attach-rule-variability**	Feature **LPG/001**	**P1**
		Availability rule **R1/001**	
		Model **M1/001**	
4	**CFG0-attach-families**	Feature **LPG/001**	**P1**
		Availability rule **R1/001**	
		Model **M1/001**	

Workflow step	Description	Attachments (Target)	Attachments (Reference)
		Family **ENG/001**	
		Model family **Models/001**	
5	**CFG0-attach-familygroups**	Feature **LPG/001**	**P1**
		Availability rule **R1/001**	
		Model **M1/001**	
		Family **ENG/001**	
		Model family **Models/001**	
		Group **Powertrain/001**	
6	**CFG0-attach-allocations**	Feature **LPG/001**	**P1**
		Availability rule **R1/001**	
		Model **M1/001**	
		Family **ENG/001**	
		Model family **Models/001**	
		Group **Powertrain/001**	
		Feature allocation **LPG/001 → P1**	
		Family allocation **ENG/001 → P1**	
		Powertrain allocation **Powertrain/001 → P1**	

Workflow process example for a configurator feature in a configurator dictionary (global variability)

Create a new workflow based on the Cfg0 Workflow template (administrator task)

1. Open Workflow Designer on Rich Client and create a new **Cfg0 Workflow** workflow process template.

2. Add the following action handlers to the **Start** and **Complete** actions of the **Do** task. Ensure the handler argument specifies a status of **TCM Released**.

   TCM Released is one of the standard status values provided by default with Teamcenter.

   The following displays both the default and configured arguments.

Action	Workflow Handler	Argument	Value
Start	**EPM-create-status**	-status	TCM Released
	CFG0-attach-constraint-rules	-attachment	target (default value)
		-configuration	Latest Working (default)
		-attachedConfigurator Context	*false*
	CFG0-attach-rule-variability	-attachment	target (default value)
		-configuration	Latest Working (default)
		-attachConfiguratorContext	false
	CFG0-attach-families	-attachment	target (default value)
		-configuration	Latest Working (default)
	CFG0-attach-familygroups	-attachment	target (default value)
		-configuration	Latest Working (default)
	CFG0-attach-allocations	-attachment	target (default value)
		-configuration	Latest Working (default)
		-attachedConfigurator Context	*false*(default value)
Complete	**EPM-set-status**	-action	append

3. Make the new workflow available to the users.

Run a new Cfg0 Workflow process on a configurator value (user task)

In this scenario, the administrator configured the **Cfg0 Workflow** process and made it available to the users.

You create data for a configurator dictionary **D1** with the following values.

Configurator Object Type	ID/Rev
Group	**Powertrain/001**
Family	**ENG/001**
Feature	**LPG/001**

Configurator Object Type	ID/Rev
Family	**TRN/001**
Feature	**M4S/001**

You create the following global include rule **R2**:

- **LPG** is included with **M4S**.

1. Select the feature **LPG/001** and choose **File→New→Workflow Process**.

   The following displays the state of the workflow process and its current attachments after each step.

   - The workflow handler **CFG0-attach-constraint-rules** is run with argument **-attachedConfiguratorContext=false**. It attaches WIP revisions for all constraint rules that reference the variability in this workflow as additional targets. In this scenario, it attaches the include rule **R2**, because the configurator rules to attach are not filtered by the configurator context items attached to this workflow.

   - The **CFG0-attach-rule-variability** workflow handler attaches the **LPG/001** and **M4S/001** features and the global rule **R2/001** as additional targets. Because of its argument **-attachConfiguratorContext=false**, no additional configurator context items are attached.

   - The **CFG0-attach-allocations** workflow handler attaches allocation WIP revisions for all configurator context items. Because of the argument **-attachedConfiguratorContext=false**, the allocation revisions to attach are not filtered by the configurator context items attached to this workflow.

Workflow step	Description	Target
1	Initiation step	Feature **LPG/001**
2	**CFG0-attach-constraint-rules**	Feature **LPG/001**
		Global include rule **R2**
3	**CFG0-attach-rule-variability**	Feature **LPG/001**
		Global include rule **R2/001**
		Feature **M4S/001**
4	**CFG0-attach-families**	Feature **LPG/001**
		Global include rule **R2/001**
		Feature **M4S/001**
		Family **ENG/001**
		Family **TRN/001**
5	**CFG0-attach-familygroups**	Feature **LPG/001**

Workflow step	Description	Target
		Global include rule **R2/001**
		Feature **M4S/001**
		Family **ENG/001**
		Family **TRN/001**
		Group **Powertrain/001**
6	**CFG0-attach-allocations**	Feature **LPG/001**
		Global include rule **R2/001**
		Feature **M4S/001**
		Family **ENG/001**
		Family **TRN/001**
		Group **Powertrain/001**
		Feature allocation **LPG/001** → **D1**
		Feature allocation **M4S/001** → **D1**
		Family allocation **ENG/001** → **D1**
		Family allocation **TRN/001** → **D1**
		Powertrain allocation **Powertrain/001** → **D1**

Workflow process example for a configurator feature in a configurator dictionary (constrained variability)

Create a new workflow based on the Cfg0 Workflow template (administrator task)

1. Open Workflow Designer on Rich Client and create a new **Cfg0 Workflow** workflow process template.

2. Add the following action handlers to the **Start** and **Complete** actions of the **Do** task. Ensure the handler argument specifies a status of **TCM Released**.

   TCM Released is one of the standard status values provided by default with Teamcenter.

   The following displays both the default and configured arguments.

Action	Workflow Handler	Argument	Value
Start	**EPM-create-status**	-status	TCM Released
	CFG0-attach-constraint-rules	-attachment	target (default value)
		-configuration	Latest Working (default)
		-attachedConfigurator Context	*true (default value)*
	CFG0-attach-rule-variability	-attachment	target (default value)
		-configuration	Latest Working (default)
		-attachConfiguratorCo ntext	false
	CFG0-attach-families	-attachment	target (default value)
		-configuration	Latest Working (default)
	CFG0-attach-familygroups	-attachment	target (default value)
		-configuration	Latest Working (default)
	CFG0-attach-allocations	-attachment	target (default value)
		-configuration	Latest Working (default)
		-attachedConfigurator Context	*true*
Complete	**EPM-set-status**	-action	append

3. Make the new workflow available to the users.

Run a new Cfg0 Workflow process on a configurator value (user task)

In this scenario, the administrator configured the **Cfg0 Workflow** process and made it available to the users.

You create data for a configurator dictionary **D1** with the following values.

Configurator Object Type	ID/Rev
Group	**Powertrain/001**
Family	**ENG/001**
Feature	**LPG/001**

Configurator Object Type	ID/Rev
Family	**TRN/001**
Feature	**M4S/001**

You create the following global include rule **R2**:

- **LPG** is included with **M4S**.

1. Select the feature **LPG/001** and choose **File→New→Workflow Process**.

   The workflow process is initiated for the feature **LPG/001** as a target and the dictionary **D1** as a reference attachment.

   The following displays the state of the workflow process and its current attachments after each step.

   - The workflow handler **CFG0-attach-constraint-rules** is run with argument **-attachedConfiguratorContext=true**. It attaches WIP revisions for constraint rules only if they apply to at least one of the configurator context items attached to this workflow, such as the include rule **R2**.

   - Workflow handler **CFG0-attach-rule-variability** attaches the **LPG/001** and **M4S/001** features and the global rule **R2/001** as additional targets. Because of its argument-**attachConfiguratorContext=false**, no additional configurator context items are attached.

   - The **CFG0-attach-allocations** workflow handler attaches allocation WIP revisions that allocate attached variability to one or more configurator context items in this workflow.

Workflow step	Description	Attachments (Target)	Attachments (Reference)
1	Initiation step	Feature **LPG/001**	**D1**
2	**CFG0-attach-constraint-rules**	Feature **LPG/001**	**D1**
		Global include rule **R2**	
3	**CFG0-attach-rule-variability**	Feature **LPG/001**	**D1**
		Global include rule **R2/001**	
		Feature **M4S/001**	
4	**CFG0-attach-families**	Feature **LPG/001**	**D1**
		Global include rule **R2/001**	
		Feature **M4S/001**	
		Family **ENG/001**	

Workflow step	Description	Attachments (Target)	Attachments (Reference)
		Family **TRN/001**	
5	**CFG0-attach-familygroups**	Feature **LPG/001**	**D1**
		Global include rule **R2/001**	
		Feature **M4S/001**	
		Family **ENG/001**	
		Family **TRN/001**	
		Group **Powertrain/001**	
6	**CFG0-attach-allocations**	Feature **LPG/001**	**D1**
		Global include rule **R2/001**	
		Feature **M4S/001**	
		Family **ENG/001**	
		Family **TRN/001**	
		Group **Powertrain/001**	
		Feature allocation **LPG/001 → D1**	
		Feature allocation **M4S/001 → D1**	
		Family allocation **ENG/001 → D1**	
		Family allocation **TRN/001 → D1**	
		Powertrain allocation **Powertrain/001 → D1**	

Workflow process example for a specific configurator rule in a dictionary

Create a new workflow based on the Cfg0 Workflow template (administrator task)

1. Open Workflow Designer on Rich Client and create a new **Cfg0 Workflow** workflow process template.

2. Add the following action handlers to the **Start** and **Complete** actions of the **Do** task. Ensure the handler argument specifies a status of **TCM Released**.

TCM Released is one of the standard status values provided by default with Teamcenter.

The following displays both the default and configured arguments.

Action	Workflow Handler	Argument	Value
Start	**EPM-create-status**	-status	TCM Released
	CFG0-attach-rule-variability	-attachment	target (default value)
		-configuration	Latest Working (default)
		-attachConfiguratorContext	false
	CFG0-attach-families	-attachment	target (default value)
		-configuration	Latest Working (default)
	CFG0-attach-familygroups	-attachment	target (default value)
		-configuration	Latest Working (default)
	CFG0-attach-allocations	-attachment	target (default value)
		-configuration	Latest Working (default)
		-attachedConfiguratorContext	*true*
Complete	**EPM-set-status**	-action	append

3. Make the new workflow available to the users.

Run a new Cfg0 Workflow process on a configurator rule (user task)

In this scenario, the administrator configured the **Cfg0 Workflow** process and made it available to the users.

You create data for a configurator dictionary **D1** with the following values.

Configurator Object Type	ID/Rev
Group	**Powertrain/001**
Family	**ENG/001**
Feature	**LPG/001**
Family	**TRN/001**
Feature	**M4S/001**

You create the following global include rule **R2**:

- **LPG** is included with **M4S**.

1. Select the feature **LPG/001** and choose **File→New→Workflow Process**.

   The workflow process is initiated for the global include rule **R2** as a target and the dictionary **D1** as a reference attachment.

   The following displays the state of the workflow process and its current attachments after each step.

   - The **CFG0-attach-rule-variability** workflow handler attaches the **LPG/001** and **M4S/001** features and the global rule **R2/001** as additional targets. Because of its argument **-attachConfiguratorContext=false**, no additional product context items are attached.

   - The **CFG0-attach-allocations** workflow handler attaches allocation WIP revisions that allocate attached variability to one or more configurator context items in this workflow.

Workflow step	Description	Attachments (Target)	Attachments (Reference)
1	Initiation step	Global include rule **R2**	**D1**
2	**CFG0-attach-rule-variability**	Global include rule **R2/001**	**D1**
		Feature **LPG/001**	
		Feature **M4S/001**	
3	**CFG0-attach-families**	Global include rule **R2/001**	**D1**
		Feature **LPG/001**	
		Feature **M4S/001**	
		Family **ENG/001**	
		Family **TRN/001**	
4	**CFG0-attach-familygroups**	Global include rule **R2/001**	**D1**
		Feature **LPG/001**	
		Feature **M4S/001**	
		Family **ENG/001**	
		Family **TRN/001**	
		Group **Powertrain/001**	
5	**CFG0-attach-allocations**	Global include rule **R2/001**	**D1**

Workflow step	Description	Attachments (Target)	Attachments (Reference)
		Feature **LPG/001**	
		Feature **M4S/001**	
		Family **ENG/001**	
		Family **TRN/001**	
		Group **Powertrain/001**	
		Feature allocation **LPG/001 → D1**	
		Feature allocation **M4S/001 → D1**	
		Family allocation **ENG/001 → D1**	
		Family allocation **TRN/001 → D1**	
		Powertrain allocation **Powertrain/001 → D1**	

Action handlers used in Product Configurator

CFG0-attach-allocations

DESCRIPTION

Attaches allocation objects that reference features, families, or groups. Such objects may be located in the target attachment or reference attachment folder. The **-configuration** argument specifies whether to attach the allocation's **Latest Working** or **Latest Released** revision.

SYNTAX

CFG0-attach-allocations
[-attachment = {target | reference}]
[-configuration = {Latest Working | Latest Released}]
[-attachedConfiguratorContext = {false | true}]
[-debug = {false | true}]

ARGUMENTS

-attachment

Attachment type with which the objects are attached to the workflow process. If any subsequent workflow handler depends on the allocation objects to be attached, either as a reference or as a target attachment, use this argument to configure two instances of this handler in the same workflow process. In such cases, the first handler is configured with **-attachment=***target***-configuration=***Latest Working* in order to attach the working revisions (if any). It is followed by the same handler configured with **-attachment=***reference* **-configuration=***Latest Released* to attach the related released objects (if any). Possible values are:

- **target**

  Allocation revisions are attached as target objects. This is the default value.

- **reference**

  Allocation revisions are attached as reference objects.

> Note:
>
> If another revision of the same configurator object thread is already attached to this workflow (either as target or reference), the handler silently skips the object. That is, the handler does not attach a second revision of the same thread.

-configuration

Specifies whether to attach the **Latest Working** or **Latest Released** revision. If any subsequent workflow handler depends on the allocation objects to be attached, either as a reference or as a target attachment, use this argument to configure two instances of this handler in the same workflow process. In such cases, the first handler is configured with **-attachment=***target***-**

configuration=*Latest Working* in order to attach the working revisions (if any). It is followed by the same handler configured with **-attachment=***reference* **-configuration=***Latest Released* to attach the related released objects (if any). Possible values are:

- **Latest Working**

  The most recently created revision with no release status is attached. This is the default value.

- **Latest Released**

  The most recently released revision is attached.

-debug

Whether or not to log status information to the syslog file. Possible values are:

- **false**

  No status information is written to the syslog file. This is the default value.

- **true**

  Status information is written to the syslog file for debugging purposes.

PLACEMENT

A typical placement is below the **EPM-create-status** action handler that creates and adds the release status to the workflow process. In many cases, it is useful to add the **CFG0-attach-allocations** handler below a **CFG0-attach-familygroups** handler.

RESTRICTIONS

None

EXAMPLES

- This example illustrates the use of the handler that attaches **Latest Working** revisions of feature, family, and group allocations for features, families, and groups in this workflow process as target attachments so that they are processed along with the variability that is already attached to the workflow. The list of allocations to add is filtered by the **Configurator Context** items attached to this workflow.

Argument	Values
-attachment	**target**
-configuration	**Latest Working**
-attachedConfiguratorContext	**true**

CFG0-attach-constraint-rules

DESCRIPTION

Attaches configurator constraint rules that reference a feature or variant option family. Such objects may be located in the target attachment or referenced attachment folder. The **-configuration** argument specifies whether to attach the **Latest Working** or **Latest Released** revision of the constraint rules.

> Note:
>
> A configurator constraint rule references the option family if the family has free-form values. Otherwise, it references the option value directly.

SYNTAX

CFG0-attach-constraint-rules

[-attachment = {target | reference}]

[-configuration = {Latest Working | Latest Released}]
[-attachedConfiguratorContext = {false | true}]

[-debug = {false | true}]

ARGUMENTS

-attachment

Attachment type with which the objects are attached to the workflow process. If any subsequent workflow handler depends on constraint rules to be attached, for example, **CFG0-attach-rule-variability**, either as a reference or as a target attachment, use this argument to configure two instances of this handler in the same workflow process. In such cases, the first handler is configured with **-attachment=***target* **-configuration=***Latest Working* in order to attach the working revisions (if any). It is followed by the same handler configured with **-attachment=***reference* **-configuration=***Latest Released* to attach the related released objects (if any). Possible values are:

- **target**

  Constraint rules are attached as target objects. This is the default value.

- **reference**

  Constraint rules are attached as reference objects.

> Note:
>
> If another revision of the same configurator object thread is already attached to this workflow (either as target or reference), the handler silently skips the object. That is, the handler does not attach a second revision of the same thread.

-configuration

Specifies whether to attach the **Latest Working** or **Latest Released** revisions. If any subsequent workflow handler depends on constraint rules to be attached, for example, **CFG0-attach-rule-variability**, either as a reference or as a target attachment, use this argument to configure two instances of this handler in the same workflow process. In such cases, the first handler is configured with **-attachment=***target* **-configuration=***Latest Working* in order to attach the working revisions (if any). It is followed by the same handler configured with **-attachment=***reference* **-configuration=***Latest Released* to attach the related released objects (if any). Possible values are:

- **Latest Working**

  The most recently created revision that does not have any release status is attached. This is the default value.

- **Latest Released**

  The most recently released revision is attached. Use this setting with care as there could be a large number of released constraint rules to attach.

-attachedConfiguratorContext

Specifies whether **Configurator Context** items that are attached to the workflow process should be used to filter constraint rules. This argument can be used as a filter to attach only constraint rules that are targeting product contexts, which are attached to the workflow process. This is useful when releasing variant features or families that are also used in constraint rules for other contexts: Filtering by configurator context prevents from accidentally attaching (and hence releasing) additional constraint rules for the configurator contexts that are not intended. Possible values are:

- **false**

  The configured revision of all constraint rules are attached, irrespective of their **Configurator Context** item scope. This is the default.

- **true**

  The configured revision of constraint rules are attached that reference a **Configurator Context** item that is attached to this workflow, for example, as a reference attachment. If no **Configurator Context** items are found to be attached to the workflow process, no additional constraint rules are added to the workflow process.

-debug

Whether or not to log status information to the syslog file. Possible values are:

- **false**

  No status information is written to the syslog file. This is the default value.

- **true**

  Status information is written to the syslog file for debugging purposes.

PLACEMENT

A typical placement is below the **EPM-create-status** action handler that creates and adds the release status to the workflow process. In many cases it is useful to add the **CFG0-attach-constraint-rules** action handler followed by a **CFG0-attach-rule-variability** action handler.

RESTRICTIONS

None

EXAMPLES

- This example illustrates the use of the handler that attaches **Latest Working** revisions of constraint rules as target attachments so that they are processed along with the values and families that are already attached to the workflow. The list of constraint rules to attach is not filtered by **Configurator Context**.

Argument	Values
-attachment	**target**
-configuration	**Latest Working**
-attachedConfiguratorContext	**false**

CFG0-attach-families

DESCRIPTION

Attaches to the workflow process variant option families that are referenced by features in the target attachment or reference attachment folder. The **-configuration** argument specifies whether to attach the **Latest Working** or **Latest Released** revisions of the variant option families.

SYNTAX

CFG0-attach-families
[-attachment = {target | reference}]
[-configuration = { Latest Working | Latest Released}]
[-debug = {false | true}]

ARGUMENTS

-attachment

Attachment type with which the objects are attached to the workflow process. Possible values are:

- **target**

  Variant option families are attached as target objects. This is the default value.

- **reference**

  Variant option families are attached as reference objects.

> Note:
>
> If another revision of the same configurator object thread is already attached to this workflow (either as target or reference), the handler silently skips the object. That is, the handler does not attach a second revision of the same thread.

-configuration

Specifies whether to attach the Latest Working or Latest Released revisions. Possible values are:

- **Latest Working**

  The most recently created revision that doesn't have any release status is attached. This is the default value.

- **Latest Released**

  The most recently released revision is attached.

-debug

Whether or not to log status information to the syslog file. Possible values are:

- **false**

  No status information is written to the syslog file. This is the default value.

- **true**

  Status information is written to the syslog file for debugging purposes.

PLACEMENT

A typical placement is below the **EPM-create-status** action handler that creates and adds the release status to the workflow process. In many cases, it is useful to add the **CFG0-attach-families** action handler between a **CFG0-attach-rule-variability** handler and a **CFG0-attach-familygroups** handler.

RESTRICTIONS

None

EXAMPLES

- This example illustrates the use of the handler that attaches **Latest Released** revisions of variant option families for the features in this workflow process as reference attachments so that they are processed along with the features that are already attached to the workflow.

Argument	Values
-attachment	**reference**
-configuration	**Latest Released**

CFG0-attach-familygroups

DESCRIPTION

Attaches to the workflow process variant option groups that reference variant option families in the target attachment or reference attachment folder. The **-configuration** argument specifies whether to attach the **Latest Working** or **Latest Released** revisions of the variant option families.

> Note:
>
> Group objects are not subject to revision rule configuration from 12.3 release. You cannot revise groups.

SYNTAX

CFG0-attach-familygroups
[-attachment = {target | reference}]
[-configuration = {Latest Working | Latest Released}]
[-debug = {false | true}]

ARGUMENTS

-attachment

Attachment type with which the objects are attached to the workflow process. Possible values are:

- **target**

  Variant option groups are attached as target objects. This is the default value.

- **reference**

  Variant option groups are attached as reference objects.

  > Note:
  >
  > If another revision of the same configurator object thread is already attached to this workflow (either as target or reference), the handler silently skips the object. That is, the handler does not attach a second revision of the same thread.

-configuration

Specifies whether to attach the **Latest Working** or **Latest Released** revisions. Possible values are:

- **Latest Working**

The most recently created revision that doesn't have any release status is attached. This is the default value.

- **Latest Released**

  The most recently released revision is attached.

-attachedConfiguratorContext

Specifies whether relevant **Configurator Context** items for which allocation objects are to be added are attached to this workflow process. This argument can be used as a filter to attach only the allocation objects that are targeting product contexts, which are attached to the workflow process. This is useful when releasing variant features, families, or groups that are allocated to multiple contexts: Filtering by configurator context prevents from accidentally attaching (and hence releasing) additional allocations to other configurator contexts that are not intended. Possible values are:

- **false**

  Configured revisions for allocations to all **Configurator Context** items will be attached. This is the default value.

- **true**

  The configured allocation revisions to attach are filtered by the **Configurator Context** items attached to this workflow. If no **Configurator Context** items are found to be attached to the workflow process, no additional allocations are added to the workflow process.

-debug

Whether or not to log status information to the syslog file. Possible values are:

- **false**

  No status information is written to the syslog file. This is the default value.

- **true**

  Status information is written to the syslog file for debugging purposes.

PLACEMENT

A typical placement is below the **EPM-create-status** action handler that creates and adds the release status to the workflow process. In many cases, it is useful to add the **CFG0-attach-familygroups** action handler between a **CFG0-attach-families** handler and a **CFG0-attach-allocations** handler.

RESTRICTIONS

None

EXAMPLES

- This example illustrates the use of the handler that attaches **Latest Released** revisions of variant option groups for the variant option families in this workflow process as reference attachments so that they are processed along with the variant option families that are already attached to the workflow.

Argument	Values
-attachment	**reference**
-configuration	**Latest Released**

CFG0-attach-rule-variability

DESCRIPTION

Attaches features and families that are referenced by a constraint rule. Such constraint rules may be located in the target attachment or reference attachment folder. The **-configuration** argument specifies whether to attach the **Latest Working** or **Latest Released** revisions of the values, families, and groups.

> Note:
>
> A configurator constraint rule references the option family if the family has free-form values. Otherwise, it references the option value directly.

SYNTAX

CFG0-attach-rule-variability
[attachment = {target | reference }]
[-configuration = {Latest Working | Latest Released }]
[-attachConfiguratorContext = {false | true }]
[-debug = { false | true }]

ARGUMENTS

-attachment

Attachment type with which the objects are attached to the workflow process. Possible values are:

- **target**

  Variant option families and values are attached as target objects. This is the default value.

- **reference**

  Variant option families and values are attached as reference objects.

> Note:
>
> If another revision of the same configurator object thread is already attached to this workflow (either as target or reference), the handler silently skips the object. That is, the handler does not attach a second revision of the same thread.

-configuration

Specifies whether to attach the **Latest Working** or **Latest Released** revisions. Possible values are:

- **Latest Working**

  The most recently created revision that has no release status is attached. This is the default value.

- **Latest Released**

  The most recently released revision is attached.

-attachConfiguratorContext

Specifies whether **Configurator Context** items that are referenced by the constraint rules in this workflow process should be attached as **reference** attachments.

> Note:
>
> The **Configurator Context** items are always added as **reference** attachments. This behavior is not affected by the **-attachment** parameter value.

Use this argument as a filter to attach only features and families that are allocated to intended product contexts.

This argument is evaluated after attaching the product context in response to the **-attachConfiguratorContext** argument (if any). The handler first attaches the product contexts of each rule as a **reference** attachment and then filters the set of features and families in each constraint rule if both **-attachConfiguratorContext** and **attachedConfiguratorContext** arguments are provided.

This is useful when releasing multi-context constraint rules. Not all variant features and families in a multi-context constraint rule are allocated to all product contexts. Use this argument when you want to release multi-context constraint rules for a specific subset of the product context scope that this constraint rule is targeting. Filtering by configurator context prevents accidentally attaching (and hence releasing) variant features and families for a configurator context that is not intended.

Options are:

- **true**

  Configurator Context items that are referenced by the constraint rules in this workflow process are attached as **reference** attachments. This is the default value.

- **false**

  No additional **Configurator Context** items are attached.

-debug

Whether or not to log status information to the syslog file. Possible values are:

- **false**

No status information is written to the syslog file. This is the default value.

- **true**

  Status information is written to the syslog file for debugging purposes.

PLACEMENT

A typical placement is below the **EPM-create-status** action handler that creates and adds the release status to the workflow process. In many cases, it is useful to add the **CFG0-attach-rule-variability** action handler between a **CFG0-attach-constraint-rules** handler and a **CFG0-attach-families** handler.

RESTRICTIONS

None

EXAMPLES

- This example illustrates the use of the handler that attaches **Latest Working** revisions of features and families that are used in the constraint rules in this workflow process as target attachments so that they are processed along with the constraint rules that are already attached to the workflow. The list of **Configurator Context** items to which the constraint rules apply are added as a reference attachments to this workflow.

Argument	Values
-attachment	**target**
-configuration	**Latest Working**
-attachConfiguratorContext	**true**

CFG0-find-constraint-conflict

DESCRIPTION

Creates the report of constraint conflicts for a given variant rule and its subtypes. The generated report is attached to the workflow process as a reference to execute this handler.

The solve profile to find the constraint conflicts are taken from the input variant rule. If no session info (solve profile) is saved on the variant rule, the system displays an error.

Similarly, the other session information such as revision rule and rule date are considered from the session information saved on the input variant rule.

If the argument values mentioned below are provided, those values override the values from session information.

The results of workflow handler are in the form of a **.json** file report with the specific schema as below:

- *TC_DATA***json\configurator\schema\CFGO_configurator_definitions.json**
- *TC_DATA***json\configurator\schema\CFGO_report_constraint_conflicts.json**

After you generate a report using the workflow handler, you can refer these schemas to get more information about the report such as constraints, severity, conflicts, session info, and variant rule name.

SYNTAX

CFG0-find-constraint-conflicts
-revisionRuleName=*revision-rule*
-ruleDate=
rule date

ARGUMENTS

-revisionRuleName

Specifies the revision rule for generating the report.

If the value is empty, then the revision rule from the input variant rule is considered.

-ruleDate

Specifies the rule date for generating the report.

The date should be in the **ISO 8601** format.

EXAMPLES

Variability data:

Family	Values
• **Engine**	• **Diesel** • **Petrol** • **Hybrid**
• **Powertrain**	• **Manual** • **Automatic**

Configurator rules:

DefaultRule D1 = (Powertrain=Manual → Engine=Diesel)

DefaultRule D2 = (Powertrain=Manual → Engine=Petrol)

Variant rule:

Variant rule VR1 = Powertrain = Manual

> Note:
> All data is configured for **Latest Working Revision Rule**.

When we start the workflow on variant rule **VR1**.

The workflow handler output report contains the conflicts between **D1** and **D2**.