

HCL Compass
Version 2.0.1

EssentialSAFE Schema



Contents

- Contents 3
- 1 Glossary of Abbreviations 5
- 2 Introduction 5
- 3 Summary of Record Types 6
- 4 Getting Started with a Populated Sample Database 7
- 5 Getting Started with an Empty Database 7
- 6 Agile Release Train 7
 - 6.1 Main 8
 - 6.1.1 Name 9
 - 6.1.2 Description 9
 - 6.1.3 Vision Statement 9
 - 6.2 Solutions 9
 - 6.2.1 Components 10
 - 6.3 Teams 11
 - 6.4 Program Increments 14
 - 6.5 Stakeholders 14
 - 6.6 Personas 15
 - 6.7 Roles 16
 - 6.7.1 Release Train Engineers 17
 - 6.7.2 System Architect 17
 - 6.7.3 Business Owners 17
 - 6.7.4 Product Managers 17
 - 6.8 NFRs 18
 - 6.9 Configure 19
 - 6.9.1 Resolutions 19
 - 6.9.2 Priorities 20
 - 6.9.3 Story Points 20
 - 6.9.4 Classes of Service 20
- 7 Timeboxes 20
 - 7.1 Program Increments 20
 - 7.1.1 Main 20
 - 7.1.2 Releases 22

7.1.3	Objectives.....	24
7.1.4	Teams.....	25
7.1.5	Iterations.....	29
7.1.6	Features.....	29
7.1.7	Notes.....	30
7.2	Iteration.....	31
7.2.1	Teams.....	31
7.2.2	Stories and Tasks.....	34
8	Work Items.....	35
8.1	Features.....	35
8.1.1	Main.....	36
8.1.2	Definition.....	38
8.1.3	Planning.....	38
8.1.4	Objectives.....	39
8.1.5	Stakeholders.....	40
8.2	Stories.....	40
8.2.1	Main.....	43
8.2.2	Definition.....	44
8.2.3	Planning.....	44
8.2.4	Tasks.....	46
8.2.5	Goals.....	46
8.3	Tasks.....	47
8.3.1	Main.....	49
8.3.2	Definition.....	50
8.3.3	Planning.....	50
8.4	Common Work Item Tabs.....	52
8.4.1	History.....	52
8.4.2	Notes.....	52
8.4.3	Subscriber List.....	52
8.4.4	Attachments.....	52
9	Queries.....	52
9.1	Public Queries.....	52
9.1.1	Viewing Scrum Notes.....	52

9.1.2	Triaging Work Items	55
9.2	Personal Queries	56
9.2.1	Kanban	58
9.2.2	Backlog	60
10	EssentialSAFe – Schema or Package?	61
10.1	EssentialSAFe as a Schema	61
10.2	EssentialSAFe as a package	61
10.2.1	Enabling a New Database	61
10.2.2	Enabling an Existing Schema	61
10.2.3	Using a Different Email, Notes or Attachment Solution	61
10.2.4	Dealing with Conflicts.....	61
10.2.5	Supplementary Packages	62
11	Permissions	62
12	Customization	63
12.1	Permission.....	63
12.2	Initialization	63
12.3	Validation.....	64
13	Best Practices.....	65

1 Glossary of Abbreviations

RT – ReleaseTrain

RTE – Release Train Engineer

PM – Product Manager

PO – Product Owner

PI – ProgramIncrement

NFR – Nonfunctional Requirement

2 Introduction

HCL Compass EssentialSAFe schema is a full featured schema which ships with HCL Compass 2.0.1 and helps team follow Essential SAFe®. Information about SAFe can be found on the [Scaled Agile Framework® website](#). SAFe and Scaled Agile Framework are registered trademarks of Scaled Agile, Inc.

The EssentialSAFe schema is ready for use out-of-the-box. Organizations can define Agile Release Trains and start using the tool quickly with minimal knowledge of schema design. Future customization of the schema is possible.

This document does not provide training for SAFe practices. Instead, it only describes how this schema can help you follow SAFe practices in your organization. For training on SAFe practices, consult the [Scaled Agile Framework website](#). There are many places in this document where we link to the Scaled Agile Framework website where you can find training and explanation of the concepts. You are encouraged to use those links to gain a greater understanding of the process.

The following assumptions are being made

- 1) HCL Compass 2.0.1 with Web Server Components should already be installed and configured.
- 2) You are familiar with administering and using HCL Compass 2.0.1 and the web server¹.
- 3) You are familiar with [Essential SAFe](#) terminology.

In this document, we will be stepping through an EssentialSAFe SAMPL database. You have two options

- 1) You can create a new sample database with all the data already in it and follow along as we describe the schema, or
- 2) You can create an empty database and add your own sample data as you follow along.

We will begin with setting up a new database, and then we will go through the record types. The order in which we go through the record types will be the most likely order in which you create them.

3 Summary of Record Types

In this section we take a quick look at the record types available in the HCL Compass EssentialSAFe schema and briefly describe how they relate to the SAFe process.

ReleaseTrain – The record that ties together an Agile Release Train and all its supporting records

Solution – A record that represents a solution that a RT delivers.

Component – A component of a solution (optional).

Team – A record that represents an agile team, with a product owner and members.

ProgramIncrement – A record that represents a Program Increment in an Agile Release Train.

TeamPI – A record to track team specific progress for a ProgramIncrement.

Iteration – A record that represents an Iteration in a Program Increment.

TeamIteration – A record to track team specific progress for an Iteration.

Release – A record to track solution versions delivered in a Program Increment

Persona – A record to assist with design thinking (optional)

¹ The EssentialSAFe schema does not work with the new web server yet. You must continue using Compass Web Server.

Feature – A work item that is completed in a Program Increment.

Story – A work item that is completed in an Iteration, which might be a child of a Feature.

Task – A work item that is completed in an Iteration, which might be the child of a Story.

4 Getting Started with a Populated Sample Database

If you would like to create a sample database with some example EssentialSAFe records, follow the instructions given in this link. In step 7 of the instructions, choose “EssentialSAFe”:

https://help.hcltechsw.com/compass/2.0.1/com.hcl.compass.doc/webhelp/oxy_ex-1/com.ibm.rational.clearquest.admin.doc/topics/t_cr_sample_db.html?hl=sample%2Cdatabase

5 Getting Started with an Empty Database

If you would like to create a new EssentialSAFe database from scratch and add the data as you go through this document, follow the steps given here- choosing EssentialSAFe as the schema for your database:

https://help.hcltechsw.com/compass/2.0.1/com.hcl.compass.doc/webhelp/oxy_ex-1/com.ibm.rational.clearquest.admin.doc/topics/t_cr_new_user_db.html

6 Agile Release Train

An [Agile Release Train](#) is the vehicle through which an organization of agile teams delivers one or more solutions to its customers and stakeholders. In HCL Compass EssentialSAFe schema this is represented the ReleaseTrain (RT) record. The user who creates an RT record will automatically be granted Release Train Engineer (RTE) role on that record and will have permission to create, modify and delete all supporting records of this RT. This applies to any other user granted the RTE or ProductManager (PM) role in the RT record. Permissions are summarized in [section 11](#).

To create a new ReleaseTrain in HCL Compass, click **New->ReleaseTrain**. If you are looking at the sample database, run the query **Public Queries->All Release Trains** and open the PIZZA release train record.

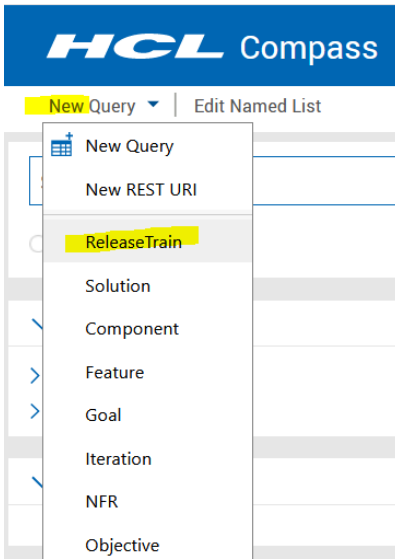


Figure 1 Creating ReleaseTrain

There are three things to keep in mind when using the **New** menu. The menu is a combo-box button. This means you can either click on the button to perform the action or click on the arrow to choose a different action.

The second thing is that the name of the menu depends on whether you have a default record type configured in your schema. If no default record type is configured (the out of the box EssentialSAFE schema does not), the name of the menu is **New Query** (such as in the figure above). Clicking on it will create a new query. If a default record type is set, then the name of the menu includes the default record type instead. For example, if you modify the schema and make Feature the default record type, then it will say **New Feature**. Clicking on it will create a new feature. In the rest of this document you will see many places where we tell you to choose a menu option like **New->Feature**. This means to choose that option from the **New** menu, whatever it is named.

The third thing is that the order of the record types you see in this menu will change as you use the product. It will remember which records you create more often and sort the list based on that. The order you see in the screenshots might not match the order you see when you are actually using the product.

6.1 Main

Let's look at the tabs on the ReleaseTrain form. For all record types, most commonly used fields will appear on the Main tab.

Welcome x *PIZZA x

ReleaseTrain:PIZZA

Main Solutions Teams Program Increments Stakeholders Personas Roles NFRs Configure

Name
PIZZA

Description
A fast and green pizza delivery service.

Vision Statement
We aim to provide a high quality pizza delivered on the greenest possible route.

Figure 2 ReleaseTrain Form

6.1.1 Name

This is how your RT and associated records will be identified. You will also see this name used to identify some of the related records, such as Program Increments and Iterations. There is no limit to the size of the name but we strongly recommend choosing a short name, under 10 characters. Choose something short but unique.

6.1.2 Description

Use this space for a longer description of your RT.

6.1.3 Vision Statement

Use this space to describe your “vision” for the solutions on this Release Train – your vision statement should answer questions like – What are our goals? What do our solutions do? Who are they for? What makes it stand out?

6.2 Solutions

The Solutions tab has a list of all solutions that are owned and delivered by this ReleaseTrain.

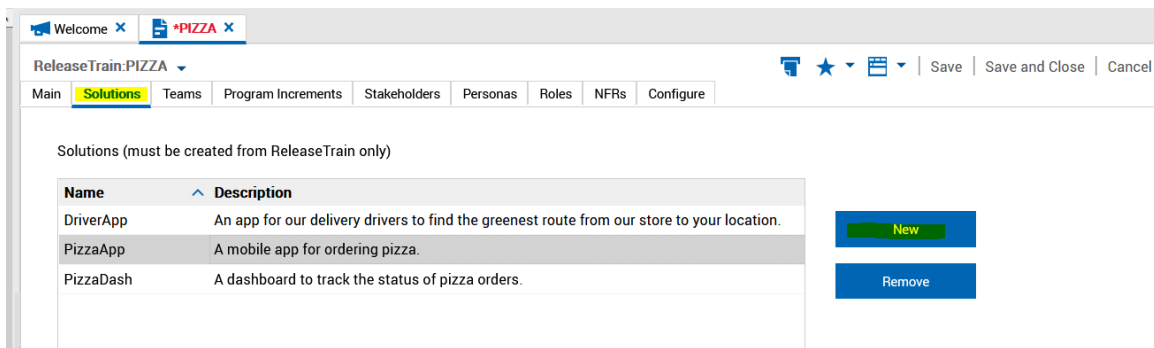


Figure 3 ReleaseTrain Solutions Tab

The RT is a vehicle through which solutions are delivered to customers and stakeholders. The HCL Compass EssentialSAFe schema has a Solution record type. At least one Solution must be created and added to your RT. A Solution belongs to the RT and must be created from the RT form. A Solution cannot be shared with multiple RTs, so you should choose a unique name for it. To create a new Solution, the RT must be in an editable state (either being submitted or modified). In the Solutions tab, click “New”. A popup Solution form will appear. Enter a name and description of your solution and click “Save”.

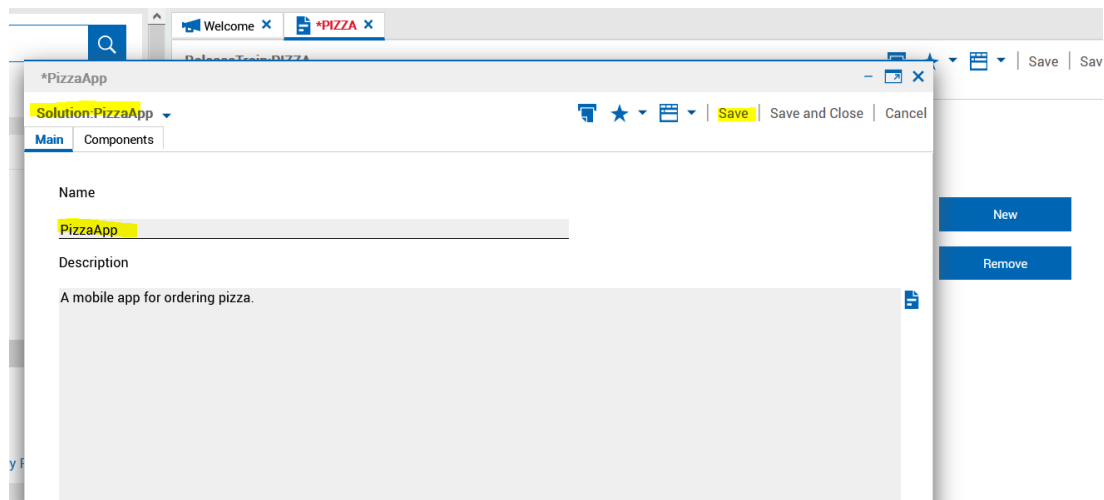


Figure 4 Solution Form Main Tab

Since you clicked “Save”, the form should stay up, allowing you to continue working with it. If you are following along with the sample database instead of creating the records, double click on any of the existing Solution records to open its form so you can follow along with the next steps.

6.2.1 Components

The Components tab on the Solution form shows all the Components of a Solution. You can further define your Solution by breaking it up into components. You can only do this when the Solution is in a non-editable state (already saved). To create a component, click **Utilities->CreateComponent**. A Component form will pop up and allow you to create a Component. The Solution field will already be set to the Solution you ran the CreateComponent from. You do not change this, but if you do the

component would get created on a different Solution. A Component needs a name. The name needs to be unique in the context of its Solution. For example, you can have two solutions, DooDad and Gadget, and they can each have a component named "UI". You cannot, however, create two "UI" components under DooDad.

If you are following along on the sample database, you can go to the Components tab and look at existing Components. Double click on one to open up its form.

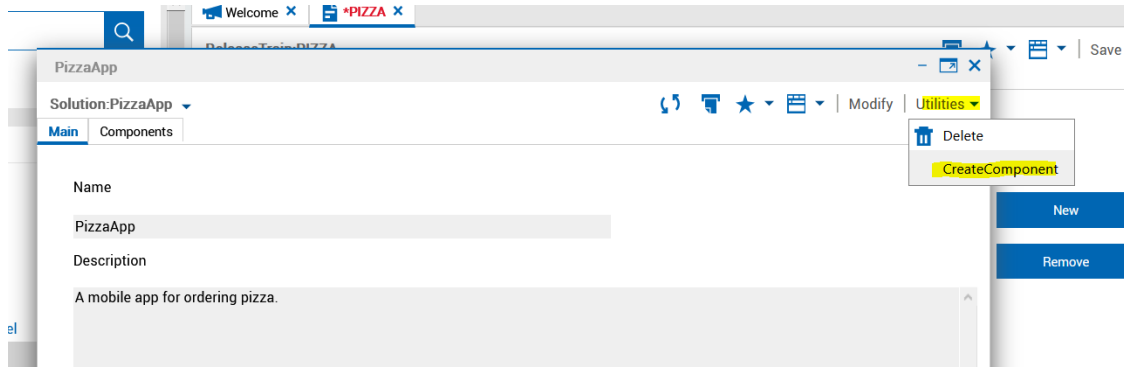


Figure 5 Creating a Component

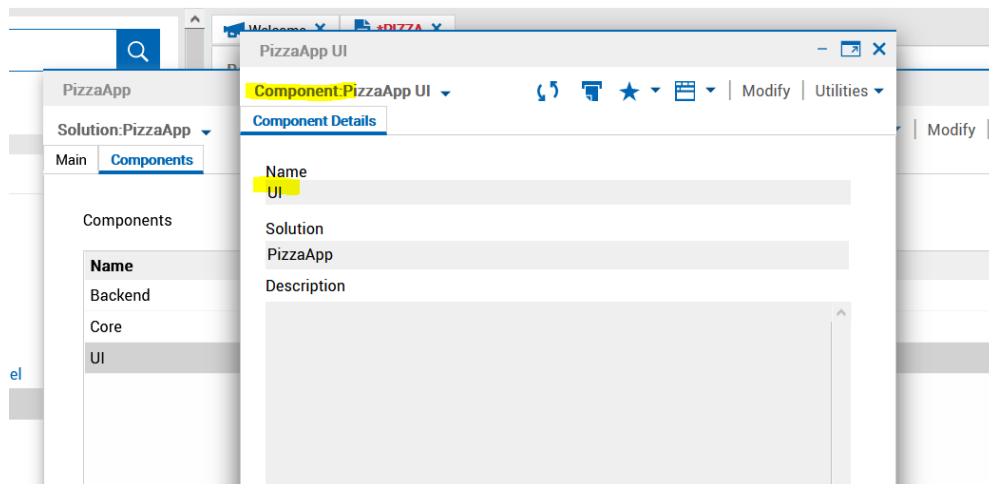


Figure 6 Component Form

6.3 Teams

The Teams tab shows all the Teams owned by this ReleaseTrain. A Release Train has an organization of agile teams working to implement the solutions that will be delivered to stakeholders and customers. The HCL Compass EssentialSAFe schema has a Team record type that represents teams in a release train. The Team record belongs to a ReleaseTrain and cannot be shared with different ReleaseTrains. The Team record includes a list of users on the team and a product owner (PO). The product owner has an important role for agile teams, as they are responsible for managing the team backlog and planning. The PO can modify the TeamPI and TeamIteration that are associated with their team. See [section 11](#) for a description of these permissions.

If you are following along with the sample database, double click on a team in the Teams tab, explore the tabs, and look at how they are set up.

To create a Team on an RT, the RT must be in a modifiable state. Click **Modify** on the RT form if you need to put it in an editable state. In the Teams tab, click **New** to add a new Team. The Team form will pop up.

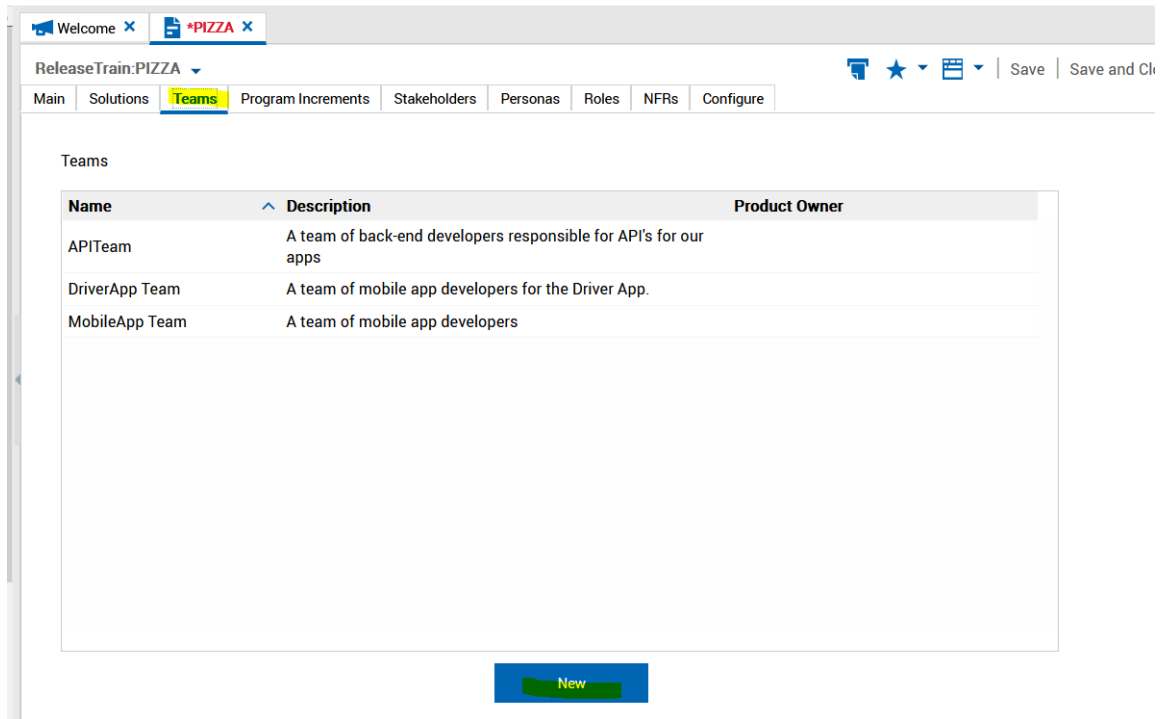


Figure 7 Creating a Team on ReleaseTrain Form

The screenshot shows the 'Main' tab of a form for a team named 'MobileApp Team'. The form contains the following fields:

- Name:** MobileApp Team
- Product Owner:** lead
- Description:** A team of mobile app developers

Figure 8 Team Form Main Tab

The screenshot shows the 'Members' tab of the 'MobileApp Team' form. It displays a table of user members and two buttons: 'Add' and 'Remove'.

login	Name	email
engineer		
lead		
QE		
user		

Figure 9 Team Form Members Tab

You do not have to add all the team members now, but do at least add a PO. The RTE or PM can always go back and modify the team members and PO. The important part is that the Teams and PO's have been defined.

To ensure the Team is associated with the correct RT, create the Team from the RT record form, as described above. Team records created any other way cannot be added to a Release Train later.

6.4 ProgramIncrements

The ProgramIncrements tab shows all the ProgramIncrement records that are owned by this ReleaseTrain. The Program Increment is one of the two timeboxes defined in HCL Compass EssentialSAFE. In this timebox the teams deliver Features that add business value to or extend the architectural runway of the solutions. Timeboxes will be covered in detail in section 7.

6.5 Stakeholders

The Stakeholders tab shows all the Stakeholder records that are associated with this ReleaseTrain. Stakeholders have a vested interest in the operation and output of the release train. These might be customers, but they could also be product managers, architects or business owners of other release trains that consume the solutions delivered by this release train. Unlike with the Team record, the RT does not own its stakeholders. They are independent records that might be added to multiple release trains in the database. You can also add stakeholders to Feature records, see section 8.1.5.

On the Stakeholders tab of the RT form, you can add existing Stakeholder records, create new Stakeholder records, and remove existing Stakeholder records. The RT must be in an editable state for you to do this.

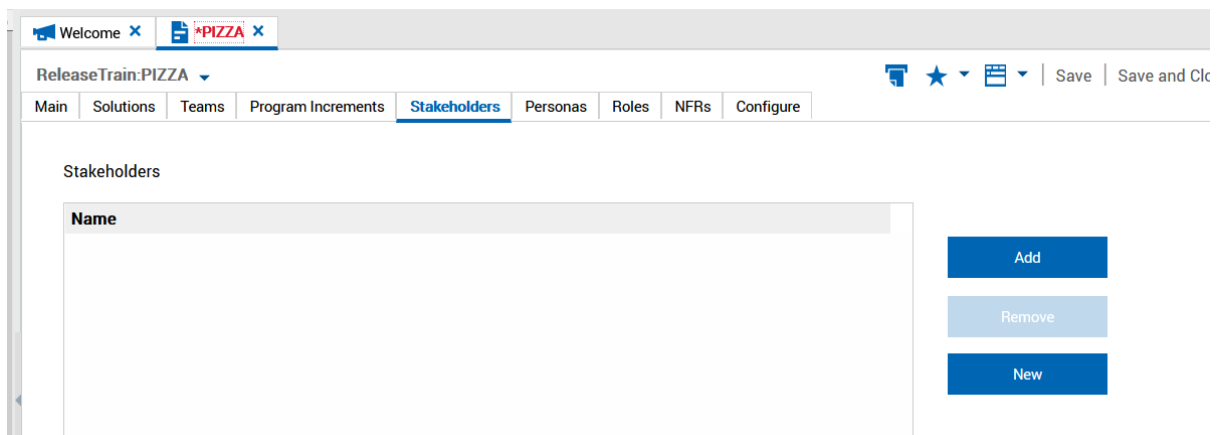


Figure 10 Stakeholders Tab

After clicking **New**, you will see the Stakeholder form. The name is required and must be unique among all stakeholder records. You can enter more details about the stakeholder in the description field.

*New Stakeholders

Stakeholder:New Record | Save | Save and Close | Cancel

Main

Name
Pizza Deliverers United

Description
PDU is an association of pizza deliverers. They have a vested interest in the DriverApp

Figure 11 Stakeholder Form

6.6 Personas

The Personas tab shows all the Persona records that are owned by this ReleaseTrain. Personas are useful for including [design thinking](#) in the development process. Personas are tied to a single ReleaseTrain record and cannot be shared with other ReleaseTrain records. When you create Story work items for this ReleaseTrain, you can choose a Persona to associate to. This can help with the design thinking activities during development.

Welcome x PIZZA x

ReleaseTrain:PIZZA

Main Solutions Teams Program Increments Stakeholders **Personas** Roles NFRs Configure

Personas

Name	Description
Ava	Ava wants to make some extra money delivering pizza.
Maritza	Maritza loves pizza. Maritza loves the beach. She would love pizza delivered to her at the beach.
Wendy	Wendy is a parent who works full time and likes to order pizza for her family on Wednesdays before she leaves work.

Delete
CreatePI
CreatePersona
CreateBacklogsAndKanbans

Figure 12 Creating a Persona from ReleaseTrain

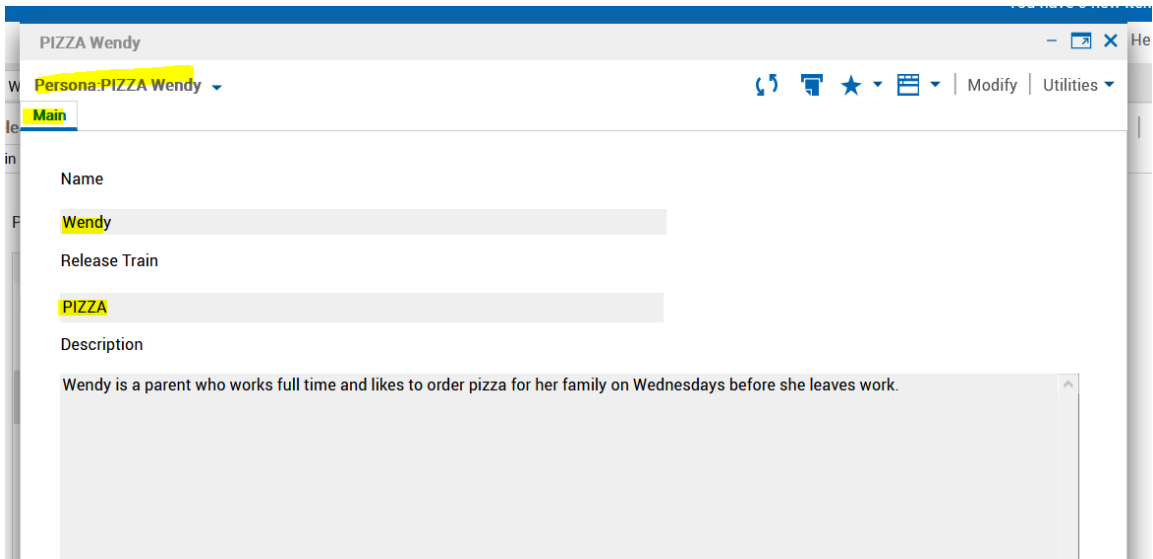


Figure 13 Creating a Persona

If you are following along with the sample database, open up some of the Personas to see what they look like.

Persona records are created using one of two ways. The **Utilities->CreatePersona** action on the ReleaseTrain form, or the New->Persona. The former method is preferred, since it will pop up a new Persona form with the ReleaseTrain already set. You only have to choose a name and type in a description. When you save and close the form, the ReleaseTrain will be automatically refreshed with the new Persona. The latter method will require you to choose the ReleaseTrain as well.

6.7 Roles

The Roles tab is where you can assign roles for users in the ReleaseTrain. The only two roles that have a functional effect on the schema are the Release Train Engineers and the Product Managers.

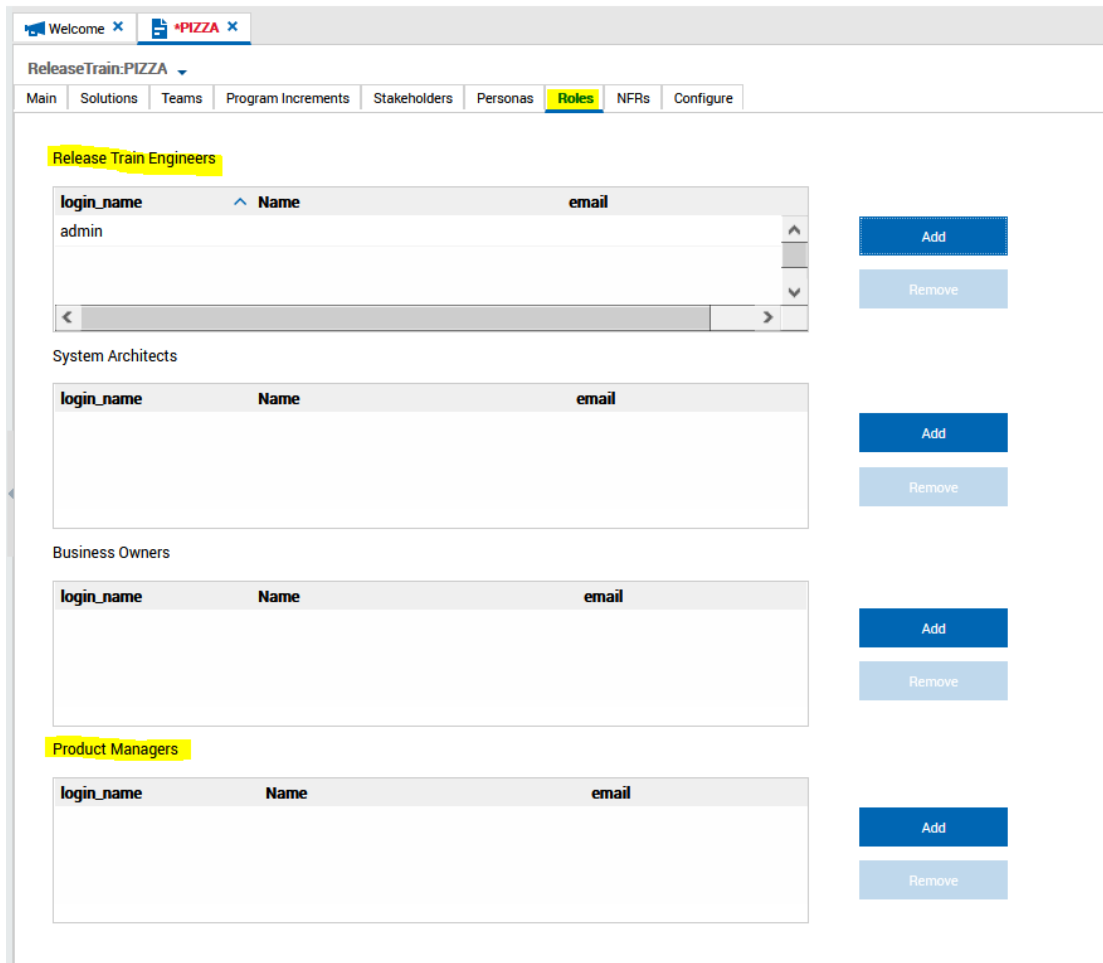


Figure 14 ReleaseTrain Roles

6.7.1 Release Train Engineers

This role has the highest privileges in the HCL Compass EssentialSAFe Release Train. The submitter of the Release Train is automatically added to this list. The RTE can delete records owned by the Release Train itself and can modify the Release Train record itself. They can create, modify, and delete all timebox related records, teams, solutions, and components. Multiple users can be defined here but there must be at least one.

6.7.2 System Architect

This role has no special privileges. You can use this to identify the system architects for your Release Train.

6.7.3 Business Owners

This role has no special privileges. You can use this to identify the Business owners for your Release Train.

6.7.4 Product Managers

Product Managers manage the Program Backlogs and Program Kanbans. They have privileges identical to Release Train Engineers.

6.8 NFRs

This tab lists the [non-functional requirements](#) (NFR) for your ReleaseTrain. NFRs describe system attributes that serve as constraints or restrictions for the design and implementation of the solutions. The program backlog and team backlogs are constrained by the NFRs.

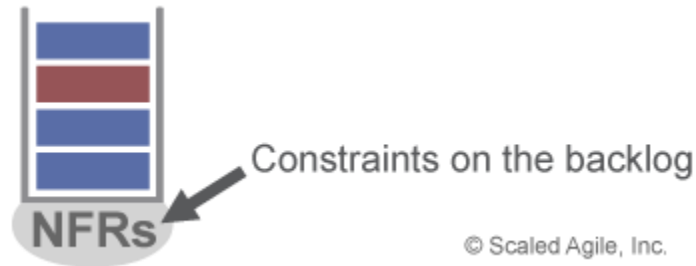


Figure 15 Backlogs are constrained by NFRs

An NFR record type exists for you to track these. The NFRs that constrain the program backlog are listed on this ReleaseTrain's NFRs tab. NFRs can also influence a team's backlog. NFRs that constrain a team's backlog are added to TeamPI records (see section 7.1.4.4) to help teams prioritize and plan their backlogs.

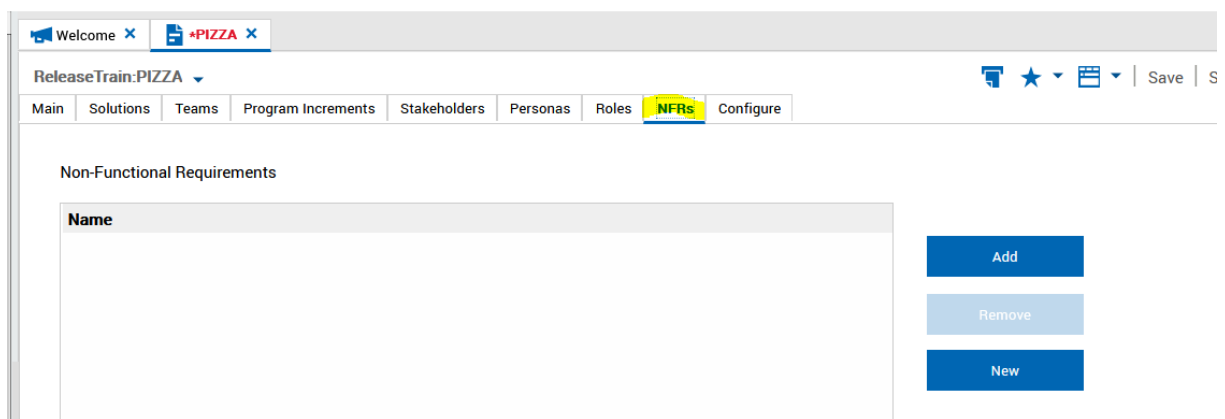


Figure 16 Nonfunctional Requirements Tab on Release Train

To add or create new NFR records directly on the ReleaseTrain, it must be in an editable state. Click **Modify** to put the ReleaseTrain in an editable state. You can also add NFRs using **New->NFR**, but in doing so, the NFR will not be tied to any ReleaseTrain or TeamPI until you **Add** it. While it may be possible to add an NFR to multiple ReleaseTrains and TeamPIs, it is recommended that you do not share NFR records. The best approach would be to always create new NFRs directly on the ReleaseTrain or the TeamPI form using the **New** button.

The NFR record type has two fields, a name and a description. More advice on creating effective NFRs can be found on the [Scaled Agile Framework website](#).

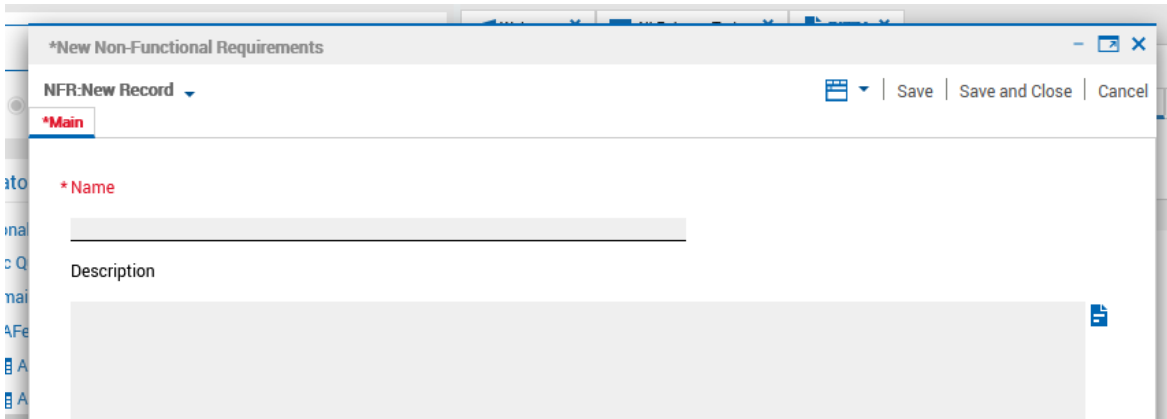


Figure 17 NFR Creation Form

6.9 Configure

Release Train Engineers can customize some of the commonly used choice lists in the tool. For example, the RTE can specify valid choices for Resolution, Priority, Story Points and Class of Service. The values you see here at record creation are recommended choices to start with. The RTE can add or remove from these lists, but they cannot be blank. The built-in queries will sort by Priority and Class of Service alphanumerically, so you should include a number in the items to ensure they are sorted the way you want.

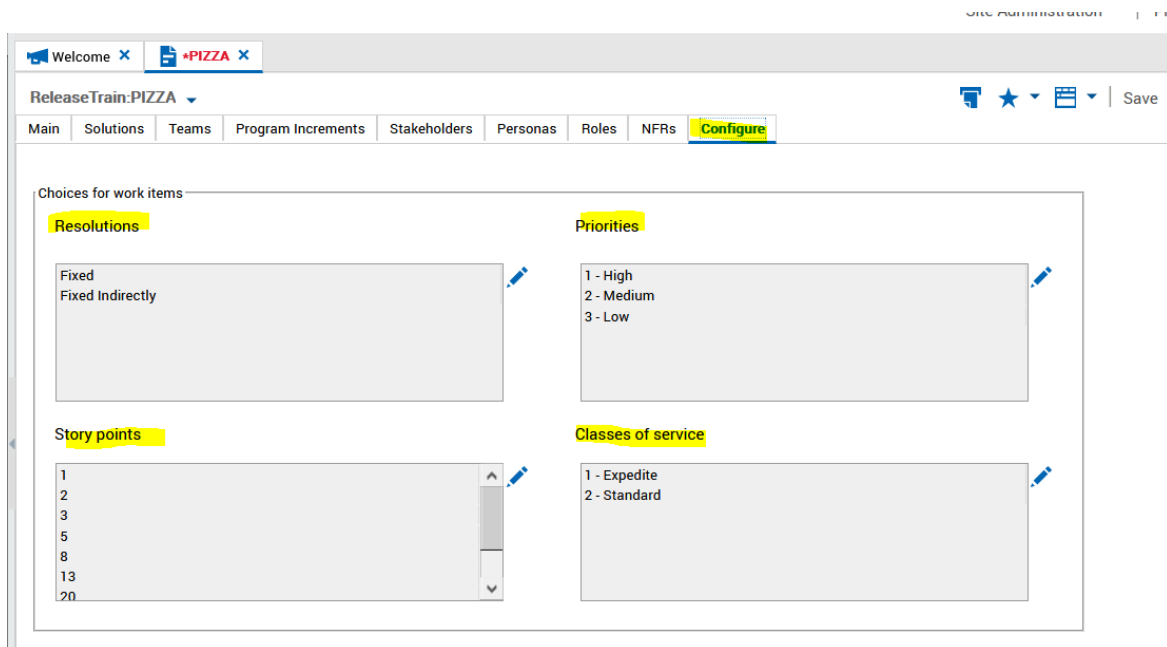


Figure 18 Configuring Release Train Choice Lists

6.9.1 Resolutions

When a feature, story or task is closed, a resolution category may be chosen. This field contains the allowable resolution category.

6.9.2 Priorities

Priority is a required field in features, stories, and tasks. These choices will be used to do Kanban prioritization planning activities. It is good practice to use a numerical value followed by a word, that way you can control how priorities are sorted.

6.9.3 Story Points

Story work items can be assigned story points for estimating the amount of effort needed. This field contains the list of allowable story points. These values must be integers.

6.9.4 Classes of Service

[Class of Service](#) is another way to optimize story execution, when **Priority** is not enough. A story, for example, may need to be expedited or have a fixed date. The default (and recommended values) are

- 0 - Fixed Date (this value is always prepended to the Classes of Service)
- 1 – Expedited
- 2 – Standard

You do not have to add “0 – Fixed Date” to the Classes of Service list. It is a built-in class of service, and it cannot be changed or removed. This class of service is used to make the **FixedDate** field mandatory on the story. You must have at least one other class of service value defined here. When adding classes of service, use a number to start, to ensure they are in the order you want in queries. For example, stories with “0 – Fixed Date”, will appear before stories with “1 – Expedited” if you sort alphanumerically.

If the ReleaseTrain has not been saved yet, save it now. Next, you will set up and configure the timeboxes.

7 Timeboxes

Solutions in a Release Train are delivered at defined intervals. These intervals are timeboxes where work gets done.

7.1 Program Increments

A [Program Increment](#) (PI) is a timebox in which Features are delivered. At the end of a PI, new releases of one or more Solutions are delivered to Stakeholders. They are typically 8-12 weeks in length.

7.1.1 Main

Program Increments can be created by Release Train Engineers and Product Managers. There are two ways to create Program Increment records. The first is on the ReleaseTrain form using **Utilities->CreatePI**. The second way is with **New->ProgramIncrement**. The former is preferred since many fields are populated automatically from the ReleaseTrain.

7.1.1.1 *Creating PI from ReleaseTrain*

From the Release Train record, click Utilities->CreatePI.

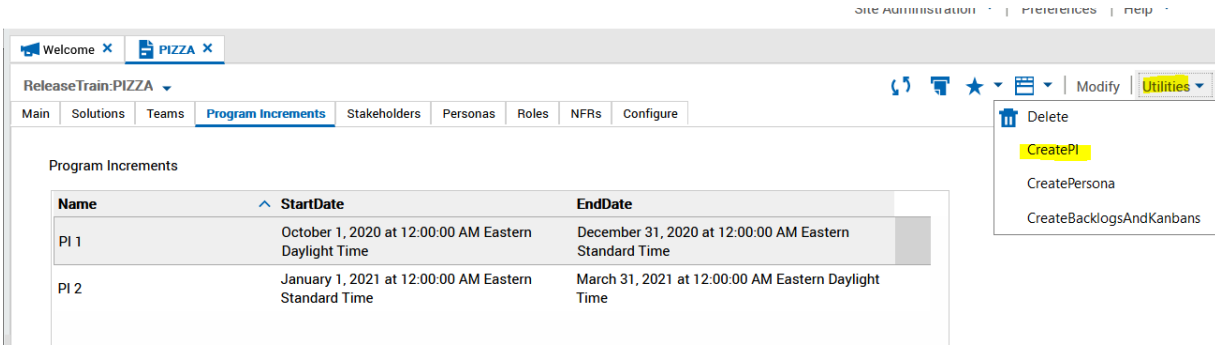


Figure 19 Choosing CreatePI from Utilities

A new Program Increment record will pop up with Release Train, Start Date and a suggested name populated. You can change the name if desired, and the name can be any length. We do recommend that you choose a short name, under 10 characters. Consider choosing names to ensure that the PIs are sorted correctly in queries. For example, PI 1 would appear before PI 2. The start date is initialized to today's date, but you can change this. Decide how long you want the PI to run for and choose an end date. A description field can hold some additional details about the Program Increment that you want to record.

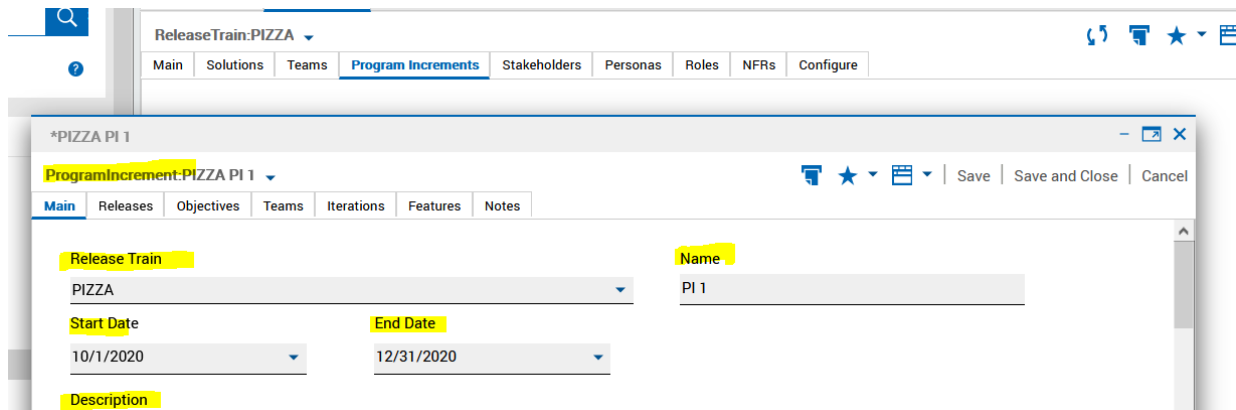


Figure 20 ProgramIncrement Form

7.1.1.2 Creating a ProgramIncrement from the New Menu

It is preferred that you create the ProgramIncrement directly from the ReleaseTrain form, but you can also create the ProgramIncrement by choosing **New->ProgramIncrement**.

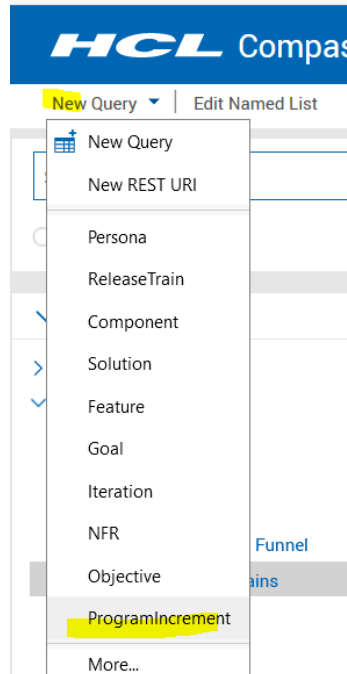


Figure 21 Creating PI Using New Menu

This will create a new ProgramIncrement with ReleaseTrain, name, and start dates not pre-filled.

Regardless of which method you choose, you must be RTE or PM for the RT to create a PI for that RT.

You can create as many Program Increments as you need. Creating them now may make it easier to create a roadmap of Features and do planning in upcoming Program Increments.

The ProgramIncrement form has other content of interest besides a name, start and end dates, and description. You can track releases for each solution, create objectives, manage team-specific status records, manage iterations and take scrum of scrum notes. Let's look at a PI now. From the Program Increment tab on the RT, open the first PI on the Release Train record.

7.1.2 Releases

A PI delivers one or more incremental releases of a solution. Each release is represented by a Release record, which lets you assign a version number of the release being delivered. You can create those releases individually or, if you have multiple solutions, all at once. You can also have multiple releases for a single solution (e.g. 1.0 and 1.0.1). The Release record helps you track which versions of the solutions get delivered in which PI.

There are several ways to create Release records. The first is manually using **New->Release**. The second is using the **Utilities->CreateRelease** action. The third is to use the **Utilities->CreateAllReleases** action. We describe the second and third methods here.

7.1.2.1 CreateRelease

With the first Program Increment record opened but in a non-editable state, run the **Utilities->CreateRelease** action. This will pop up a Release creation window with RT and PI already set in the

form to match the PI you ran the action on. You only need to set a solution and version number. You can add a description if you like, but it is not required.

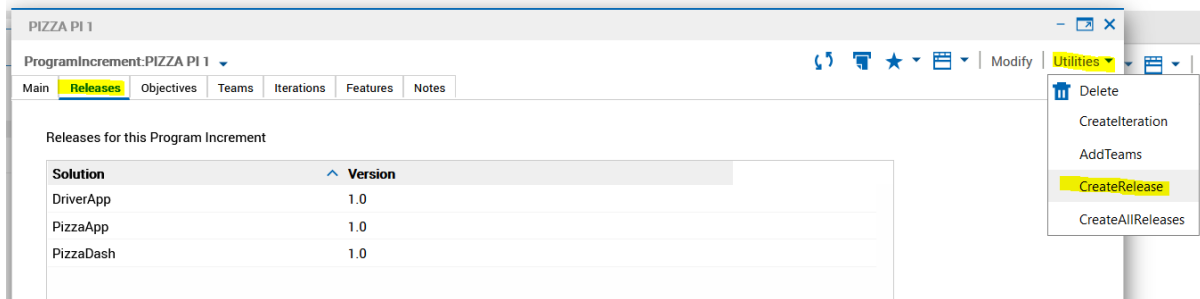


Figure 22 Creating Release from ProgramIncrement Form

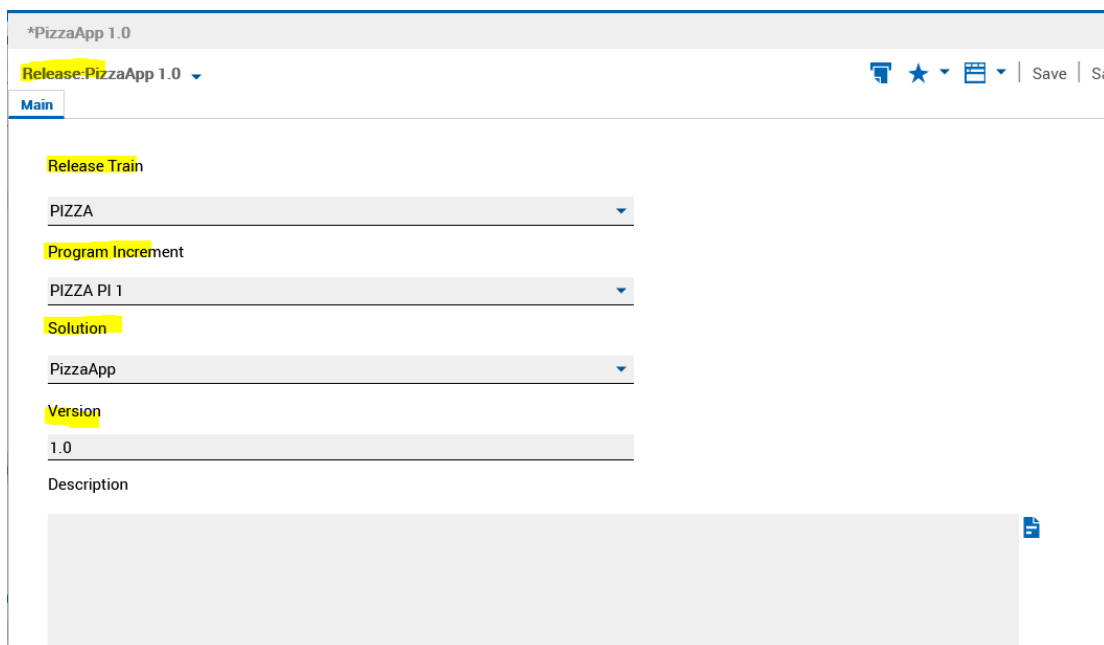


Figure 23 Release Creation Form

7.1.2.2 CreateAllReleases

If there are multiple solutions in this RT, it may be easier to create multiple Release records at one time. You can do this if all have the same version number (e.g. 1.0). To do this, you need to set “Default version for releases in this Program Increment” on the PI. This is set on the Releases tab. Now you can run the **Utilities->CreateAllReleases** action. This will automatically create a Release record for every solution using the version specified as the default. This action is smart enough to detect if a Release already exists and it will skip it. So, if you created one Release manually, you could still run **CreateAllReleases** to create the rest.

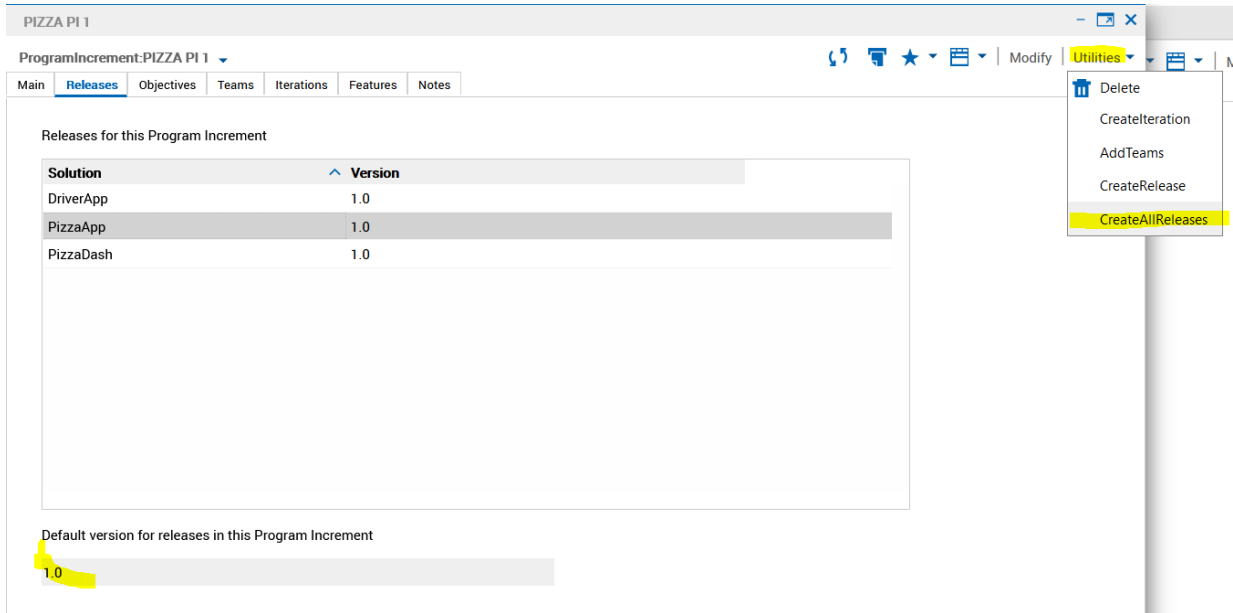


Figure 24 Running CreateAllReleases Action

7.1.3 Objectives

The RTEs and PMs can create organizational [Objectives](#) which summarize the teams' individual objectives for the PI (see section [Program Increment](#)), or define organization-wide objectives for the PI. The objective is represented by an Objective record, which lets you specify a headline, business value, description and commitment. The best way to create PI objectives is directly on the Objectives using the **New** button, because this will automatically link the newly created Objective to the PI.

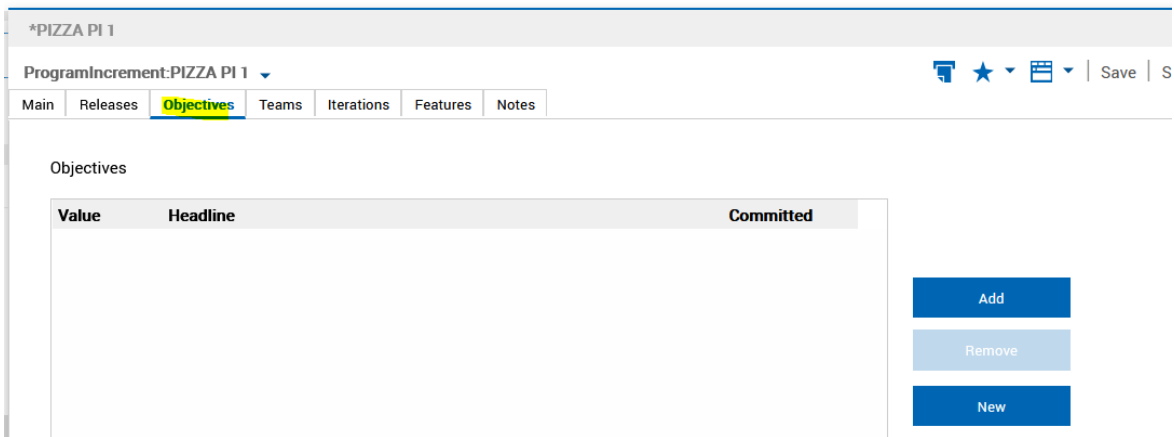


Figure 25 Objectives Tab on ProgramIncrement

You will need to **Modify** the PI record to use the **New** button. Clicking the New button pop up an Objective creation form. You must choose a Headline and Business Value. Optionally you can add a description. Consider writing [SMART](#) Objectives that are specific, measurable, achievable, realistic and time-bound. The Objective also has a committed value which will can be used to distinguish between

Committed and Uncommitted objectives. When done creating the Objective, click **Save and Close**. The Objective lists will be updated with the newly created Objective.

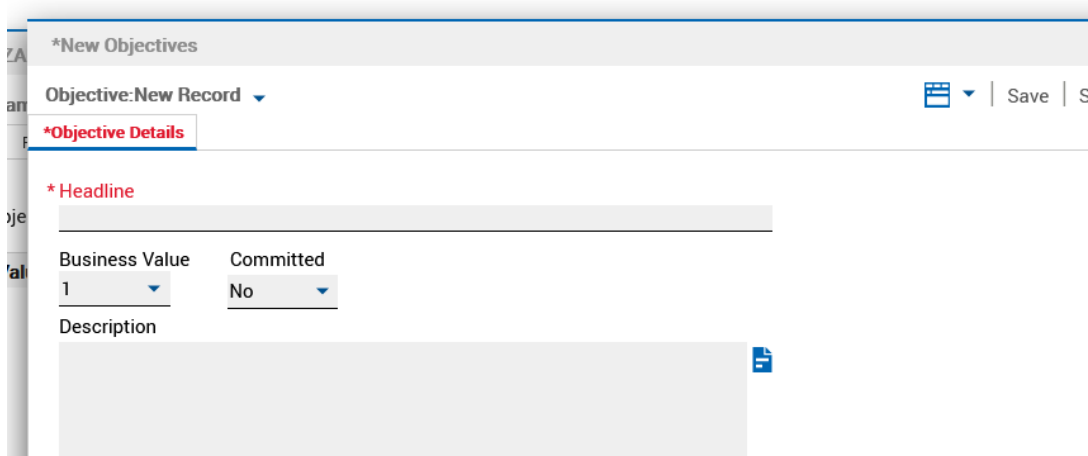


Figure 26 Objective Creation Form

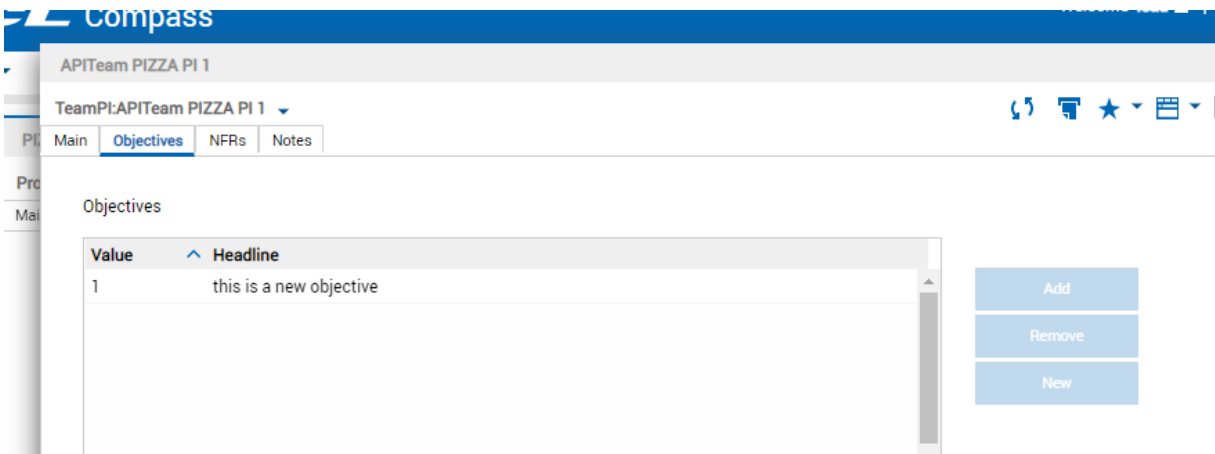


Figure 27 Newly Created Objective on ProgramIncrement

Objectives can be created by anyone from **New->Objectives** menu, however they can only be added to a PI by an RTE or PM, or to a TeamPI by an RTE, PM or PO.

While you can **Add** existing Objective records, you should use caution when sharing Objective records between different ReleaseTrain records. It may also be undesirable to share Objective records between different PIs. The reason for this is an objective might be committed in one ReleaseTrain or PI, but not in the other. The best practice is to create a new Objective every time, and in the headline you may want to identify which ReleaseTrain or PI it should be for, e.g. "PIZZA/API Team – APIs should comply with FIP140-2 security standards".

After creating objectives, you must save the PI. You should click **Save** now (not **Save and Close**, since we are not finished with the PI).

7.1.4 Teams

When the ProgramIncrement record is created, it will also create a TeamPI record for every team defined in the ReleaseTrain. The TeamPI record tracks team-specific information related to the PI, such

as scrum notes and team objectives. The Teams tab will list all TeamPI records created for this PI, as shown below.

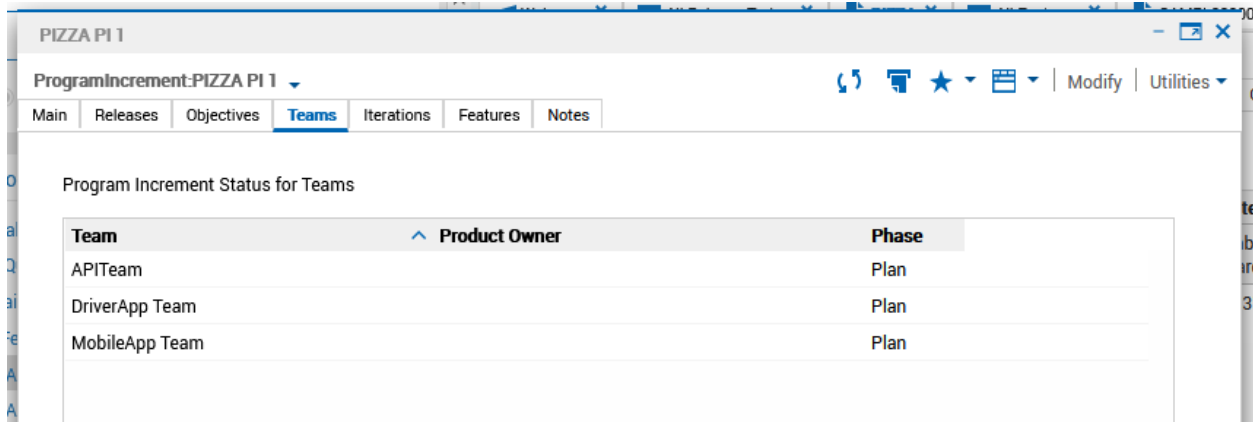


Figure 28 Teams Tab on ProgramIncrement

If you add a Team to the ReleaseTrain after the PI was created, the TeamPI will not automatically get created for the new Team. You can add it to the PI while the PI is in a non-editable state using the **Utilities->AddTeams** action. This action will look for new teams and add any missing TeamPI records. We will now describe more about the TeamPI record.

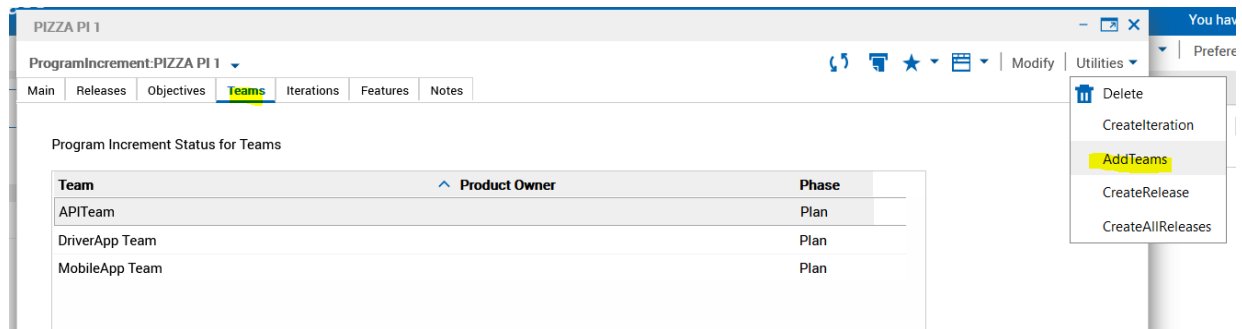


Figure 29 Running AddTeams Action on ProgramIncrement

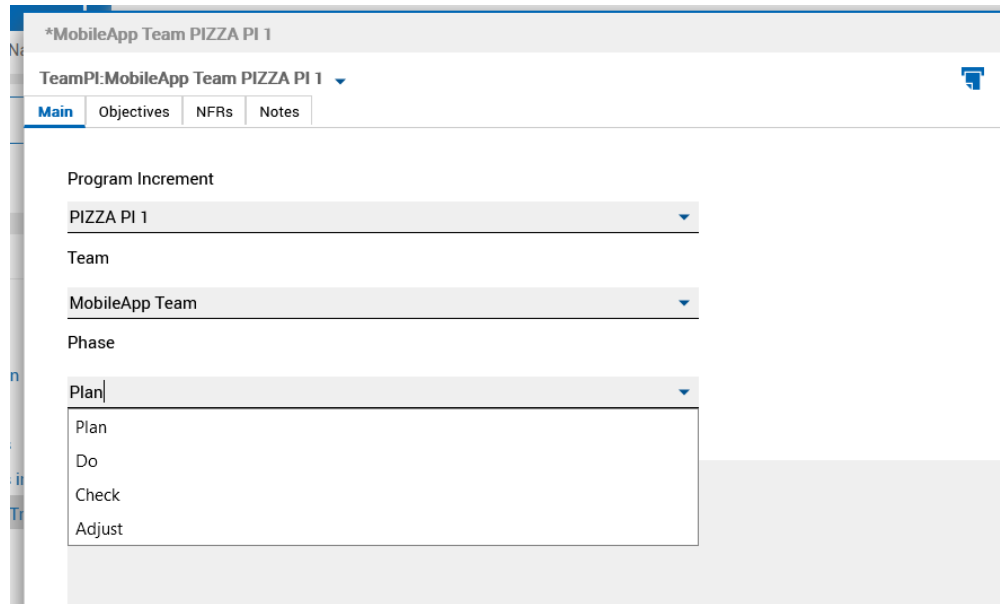
7.1.4.1 TeamPI

The TeamPI record is a special record that allows individual agile teams to track their own progress, objectives and NFRs as they work through the Program Increment. The easiest way to get to these records is through the Teams tab on the ProgramIncrement. In a typical Agile Release Train, the product owners of the teams collect information about the team's progress in this TeamPI record and bring it back to the product manager for the weekly scrum of scrum status meeting.

TeamPI records are created automatically, as described in section 7.1.4, but they can be modified by the product owner of the associated team. The TeamPI tabs are described below.

7.1.4.2 Main

If the TeamPI was automatically created through earlier steps, then PI and Team will have already been set and you should not change these. You can optionally track the [Shrewhart Plan Do Check Adjust cycle](#) for the team during the PI. Simply set the Phase as appropriate (as in Figure 30). The team's phase will show up in the Teams tab on the ProgramIncrement (see Figure 29). In the Main tab you can also provide a description. This could be a summary statement about what the team hopes to accomplish in this PI.



The screenshot shows a web interface for a TeamPI record. At the top, there is a title bar with the text '*MobileApp Team PIZZA PI 1'. Below this is a breadcrumb-style header 'TeamPI: MobileApp Team PIZZA PI 1' with a dropdown arrow. A navigation bar contains four tabs: 'Main' (which is active and highlighted in blue), 'Objectives', 'NFRs', and 'Notes'. The main content area is divided into several sections, each with a label and a dropdown menu: 'Program Increment' with 'PIZZA PI 1' selected; 'Team' with 'MobileApp Team' selected; and 'Phase' with 'Plan' selected. A dropdown menu is open for the 'Phase' field, showing the options 'Plan', 'Do', 'Check', and 'Adjust'. The interface is clean and uses a light gray color scheme.

Figure 30 TeamPI Plan Do Check Adjust Cycle

7.1.4.3 Objectives

The Objectives tab shows a list of team objectives for the PI. During PI Planning, Teams create [objectives](#) they wish to meet during the PI. Objectives also help the team in goal setting when doing iteration planning and working on stories. This is where they will create and add their PI objectives. How you create team PI objectives on the TeamPI form is similar to how you create PI objectives on the PI form. First put the TeamPI in an editable state and then click **Add** or **New**. Refer to section 7.1.3 for more about the Objective record. Teams usually create objectives first in the TeamPI records, and then the RTE will summarize these objectives in the PI record and sometimes add new objectives for the entire organization. Different teams may share the same objective, especially if more than one team works on a single Feature. For a shared Objective, one team can create it and the other teams can add it. If you are creating a new Objective and not sharing it with other teams, you may want to indicate this in the headline. For example, set the headline to “PIZZA/APITeam – APIs should comply with FIP140-2 security standards”. For a shared Objective, don’t put a team in the headline to indicate it can be shared, e.g. “PIZZA – Order a Pizza through the mobile app.”

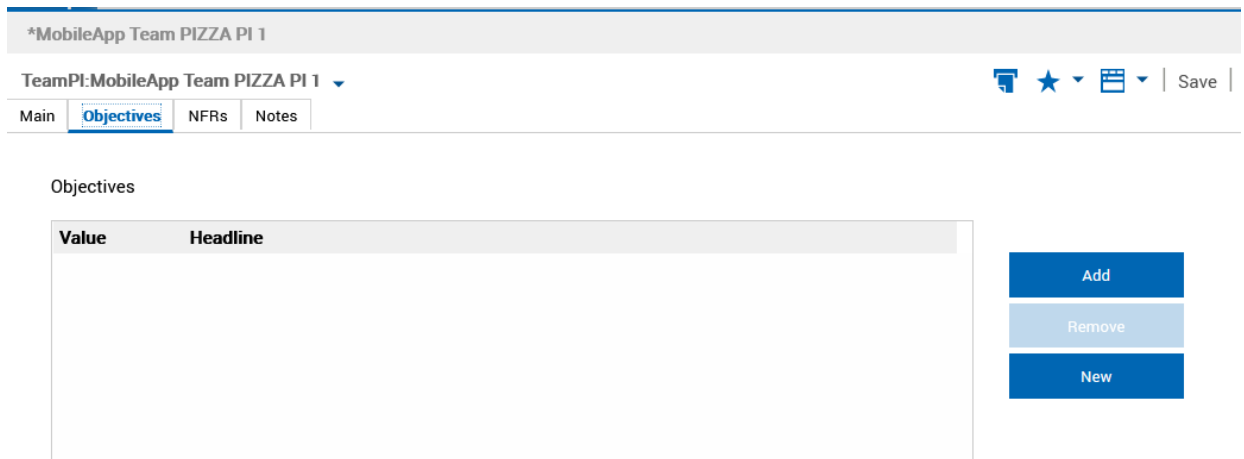


Figure 31 Objectives on TeamPI Form

7.1.4.4 NFRs

The NFRs tab lists non-functional requirements a team's backlog. For more information about NFRs, see section 6.8.

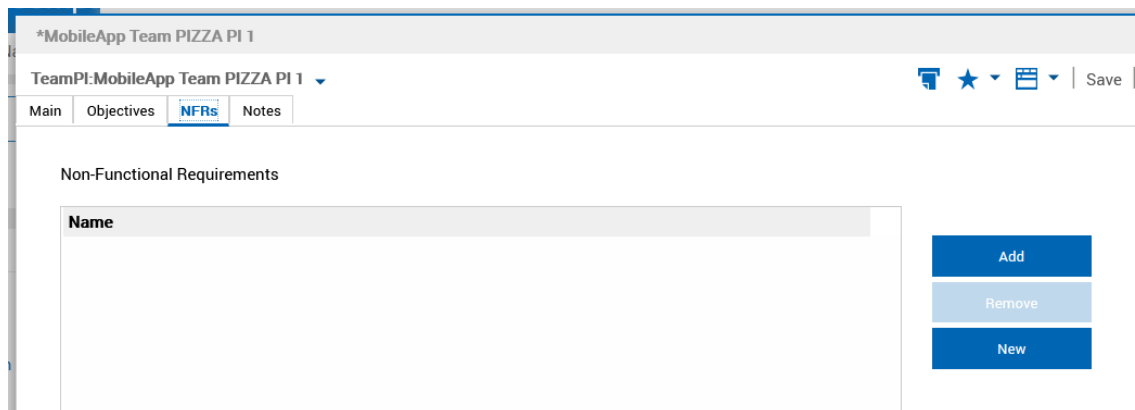


Figure 32 NFRs Tab on TeamPI

7.1.4.5 Notes

The Notes tab is where Product Owners can take daily team scrum notes, or notes of any kind, throughout the life of the PI.

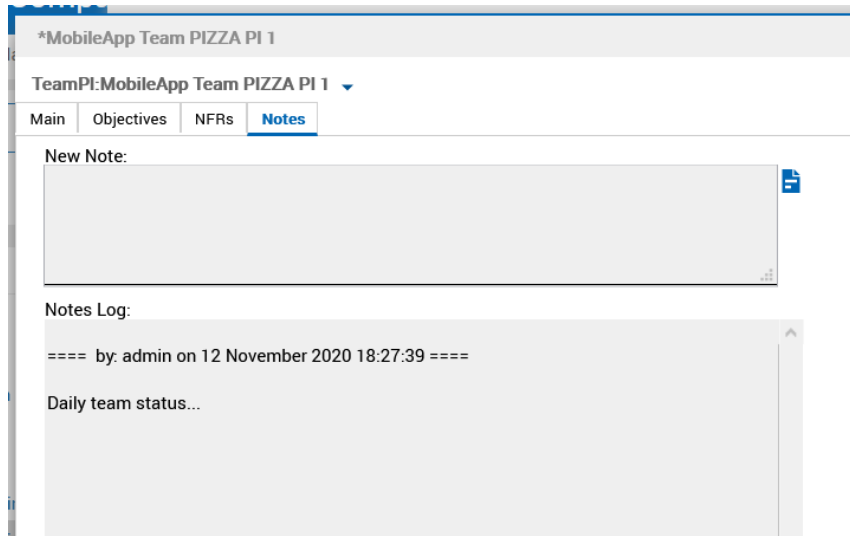


Figure 33 Notes Tab on TeamPI

7.1.5 Iterations

A Program Increment is made up of iterations. [Iterations](#) are smaller timeboxes of 2-4 weeks in length during which Story and Task work items are completed. The Iterations tab on PI lists all the Iteration records for this ProgramIncrement. You will create your Iterations from the PI record itself. Creation of the Iteration records will be covered in section 7.2.

Name	StartDate	EndDate
Iter 1	October 1, 2020 at 12:00:00 AM Eastern Daylight Time	October 15, 2020 at 12:00:00 AM Eastern Daylight Time
Iter 2	October 16, 2020 at 12:00:00 AM Eastern Daylight Time	October 31, 2020 at 12:00:00 AM Eastern Daylight Time
Iter 3	November 1, 2020 at 12:00:00 AM Eastern Daylight Time	November 15, 2020 at 12:00:00 AM Eastern Standard Time
Iter 4	November 16, 2020 at 12:00:00 AM Eastern Standard Time	November 30, 2020 at 12:00:00 AM Eastern Standard Time
Iter 5	December 1, 2020 at 12:00:00 AM Eastern Standard Time	October 15, 2020 at 12:00:00 AM Eastern Daylight Time
Iter 6	December 16, 2020 at 12:00:00 AM Eastern Standard Time	October 31, 2020 at 12:00:00 AM Eastern Daylight Time

Figure 34 Iterations Tab on ProgramIncrement

7.1.6 Features

The Features tab shows you all the Feature records which have been scheduled for this PI. A Feature will automatically show up here when its PI is set to this one (such as in Figure 36).

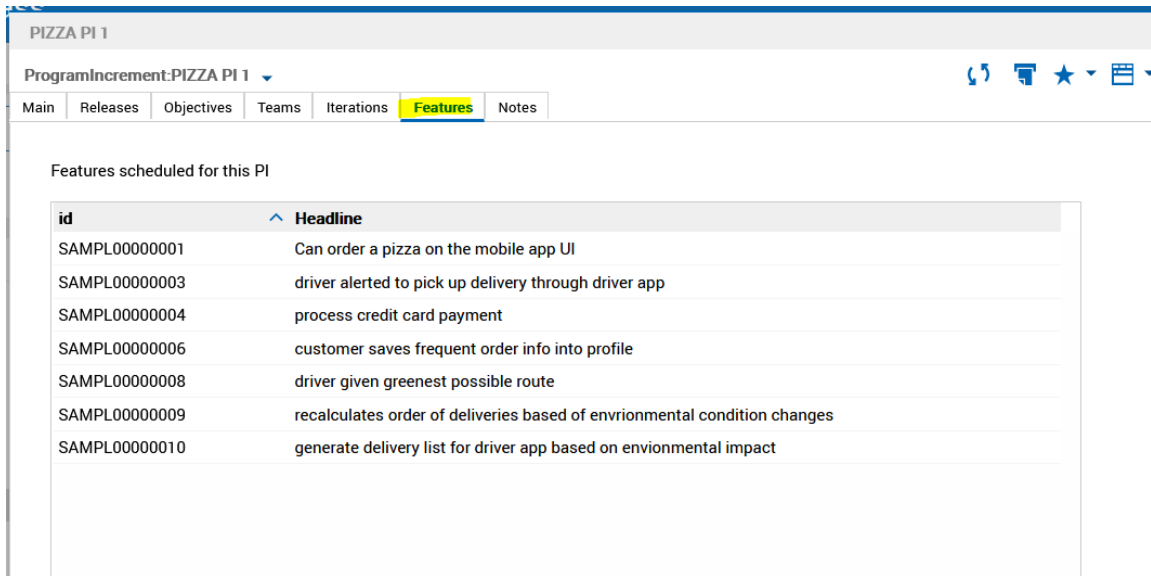


Figure 35 Features Tab on ProgramIncrement

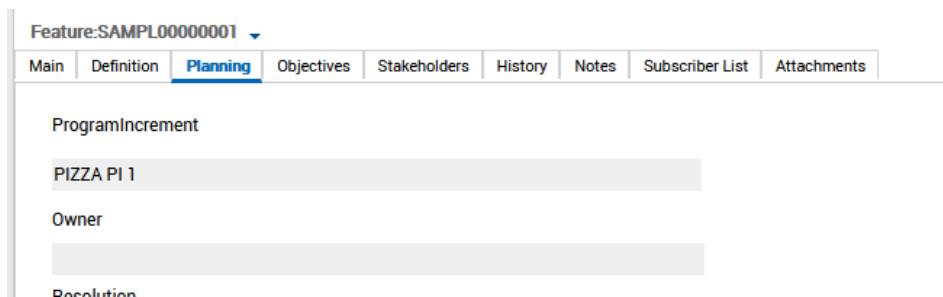


Figure 36 A Feature Set to Program Increment PI 1 on PIZZA ReleaseTrain

7.1.7 Notes

The Notes tab is a space for a Product Manager or Release Train Engineer to take scrum of scrum notes.

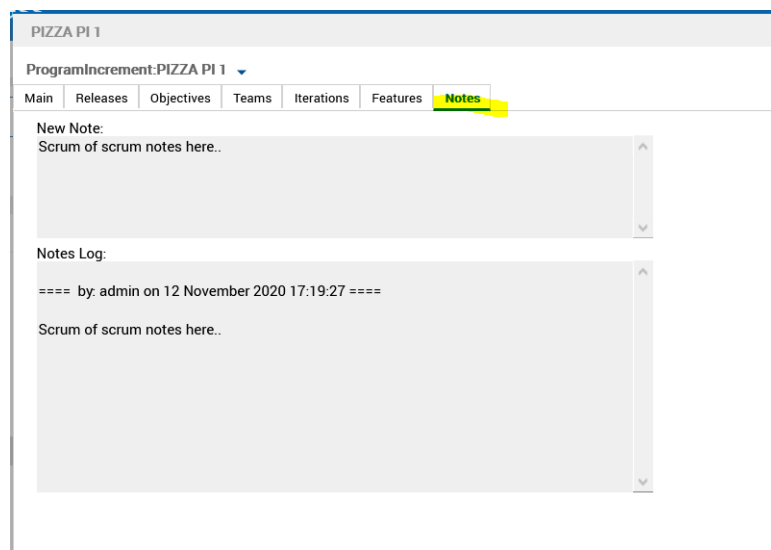


Figure 37 Notes Tab on ProgramIncrement

7.2 Iteration

A Program Increment is made up of iterations. [Iterations](#) are the shortest timeboxes and are where stories are completed. They typically last 2-4 weeks depending on organizational needs. An Iteration record tracks the iteration. They can only be created and modified by RTEs and PMs.

You can create Iterations on a ProgramIncrement record when it is in a non-editable state. Run the **Utilities->CreateIteration** and a new Iteration form will pop up pre-populated with the PI, a suggested name, and a start date (see Figure 39). Consider choosing a name that will sort iterations alphanumerically in the proper order (e.g. Iter 1, Iter 2). We strongly recommend choosing a short name, preferably under 10 characters.

You must specify an end date before saving. You can create as many Iterations as you need for this PI. You can create them all up-front so that teams can do [planning](#) across multiple iterations, or you can create them as you need them and only plan for the next iteration.

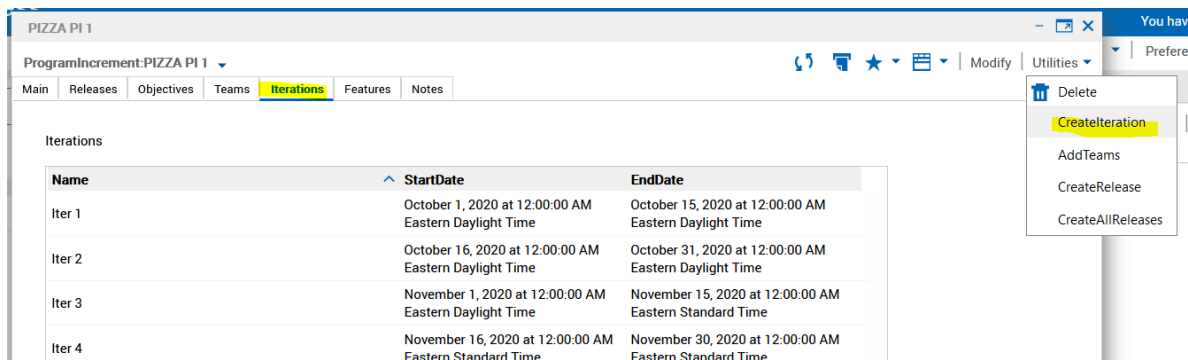


Figure 38 CreateIteration Action on ProgramIncrement

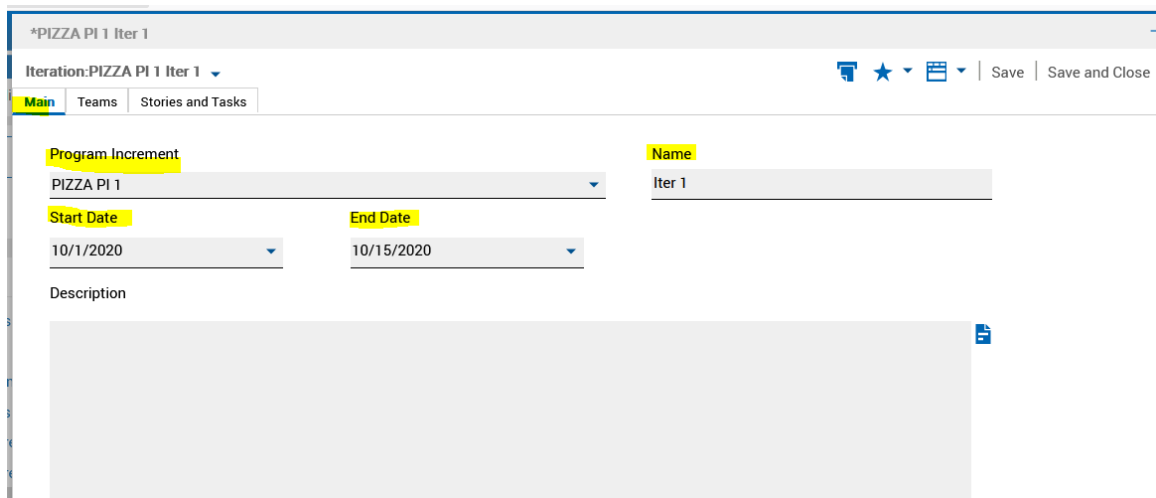


Figure 39 Creating Iteration

7.2.1 Teams

As with creating a PI, saving a new Iteration will trigger the creation of a TeamIteration record for each team in the ReleaseTrain. The TeamIteration is to an Iteration as the TeamPI is to a ProgramIncrement. The TeamIteration records capture team specific information related to the Iteration, such as the team

phase, goals and retrospective. The TeamIteration records will not be created until after you save the new Iteration.

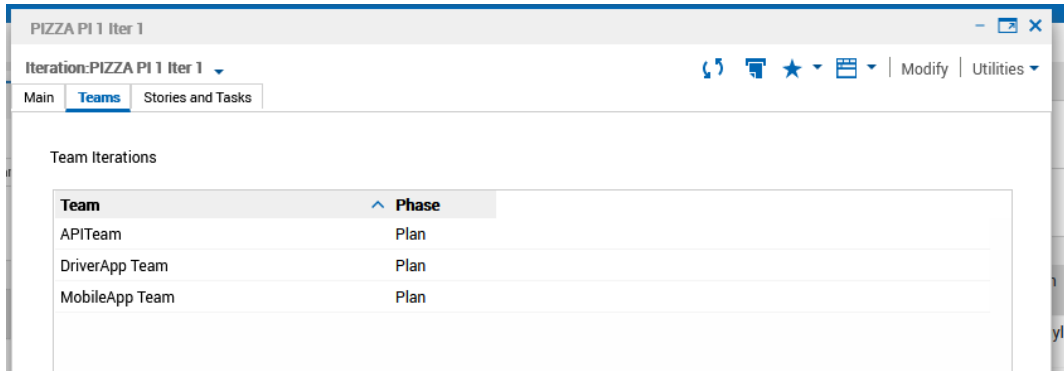


Figure 40 Teams Tab on Iteration

If you add a Team to the ReleaseTrain after the Iteration was created, the TeamIteration will not automatically be created for the new Team. An RTE or PM can add new teams to the Iteration. While the Iteration is in a non-editable state, run the **Utilities->AddTeams** action. This action will look for new teams and add any missing TeamIteration records to the Iteration.

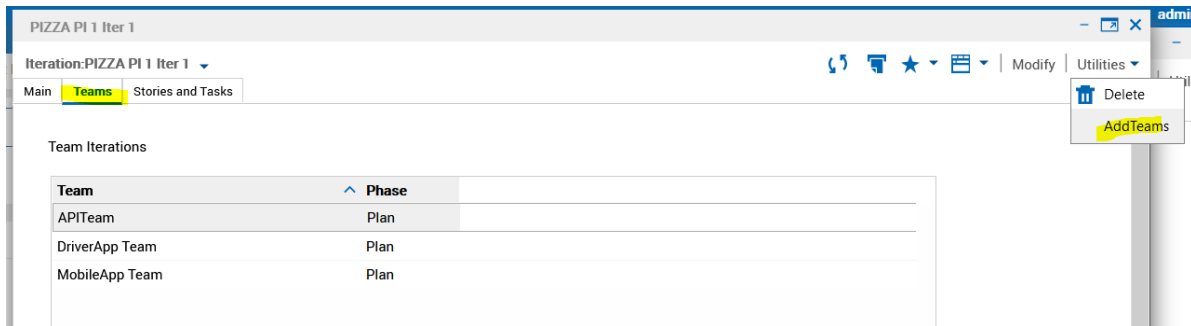


Figure 41 Running AddTeams on Iteration

7.2.1.1 TeamIteration

Now let's look at a TeamIteration. Double click on one of the records listed on the Teams tab.

7.2.1.1.1 Main

The Iteration and Team values on this tab were automatically set during Iteration creation or when you ran the **AddTeams** action. You can also track the [PDCA cycle](#) (Plan-Do-Check-Act) here. There is a space for a longer description of what your team is doing with the iteration.

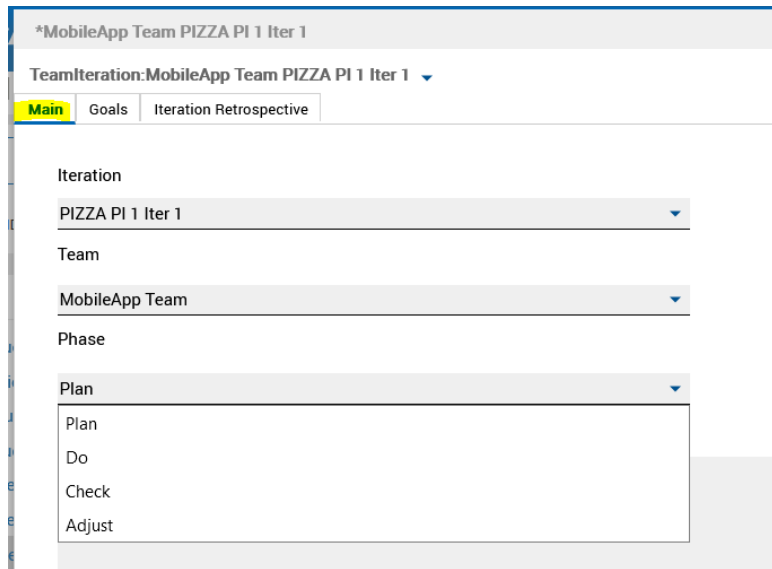


Figure 42 TeamIteration Main Tab

7.2.1.1.2 Goals

Agile Teams can enter [goals](#) for the iteration which should align with the team and organizational objectives put forth by the PI planning. A new goal includes a headline and description and includes one or more objectives (see Figure 43). Goals can be entered here, or from the **New-> Record** menu and added here later. You can also add one or more Goal records to a Story. This can help align your work with the Iteration goals. You will learn more about this in section 8.2.5.

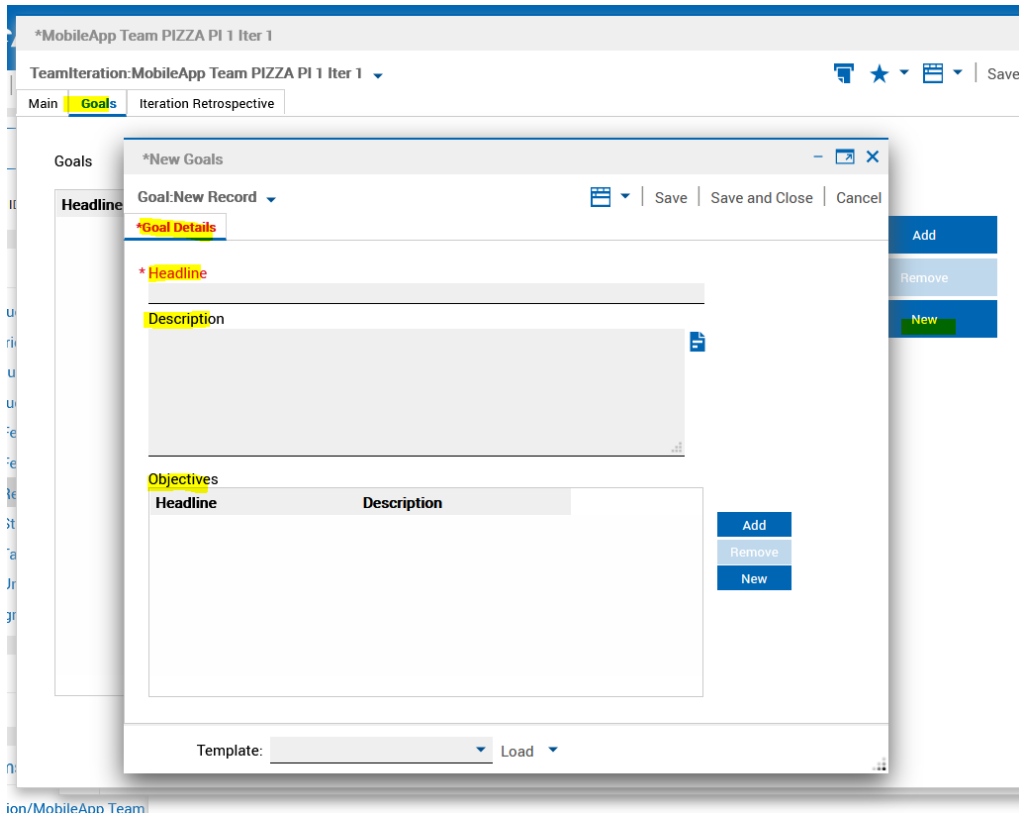


Figure 43 Creating a Goal on TeamIteration

7.2.1.1.3 Iteration Retrospective

When the iteration ends, the PO can conclude the iteration with a [retrospective](#). What went right, what went wrong? What can be done better? What unfinished work should go into the next iteration? What defects were found? The output of this retrospective can be captured here.

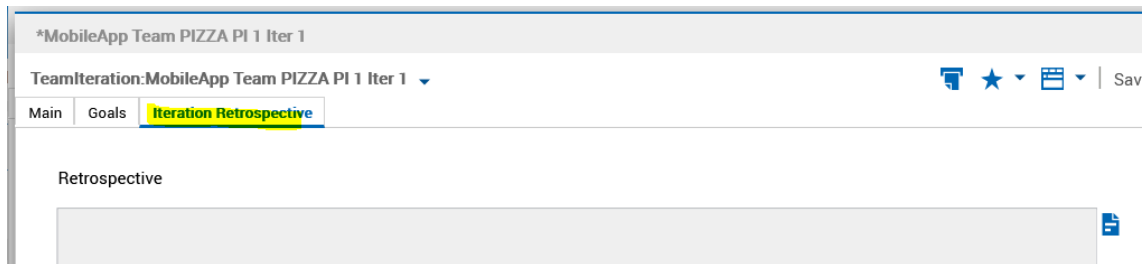


Figure 44 Iteration Retrospective Tab

7.2.2 Stories and Tasks

This tab shows all the Stories and Tasks scheduled for this iteration, the teams responsible for them, and their current state. It is a quick way to see where all the teams are in terms of completing their work. There is no action within the Iteration record for this tab.

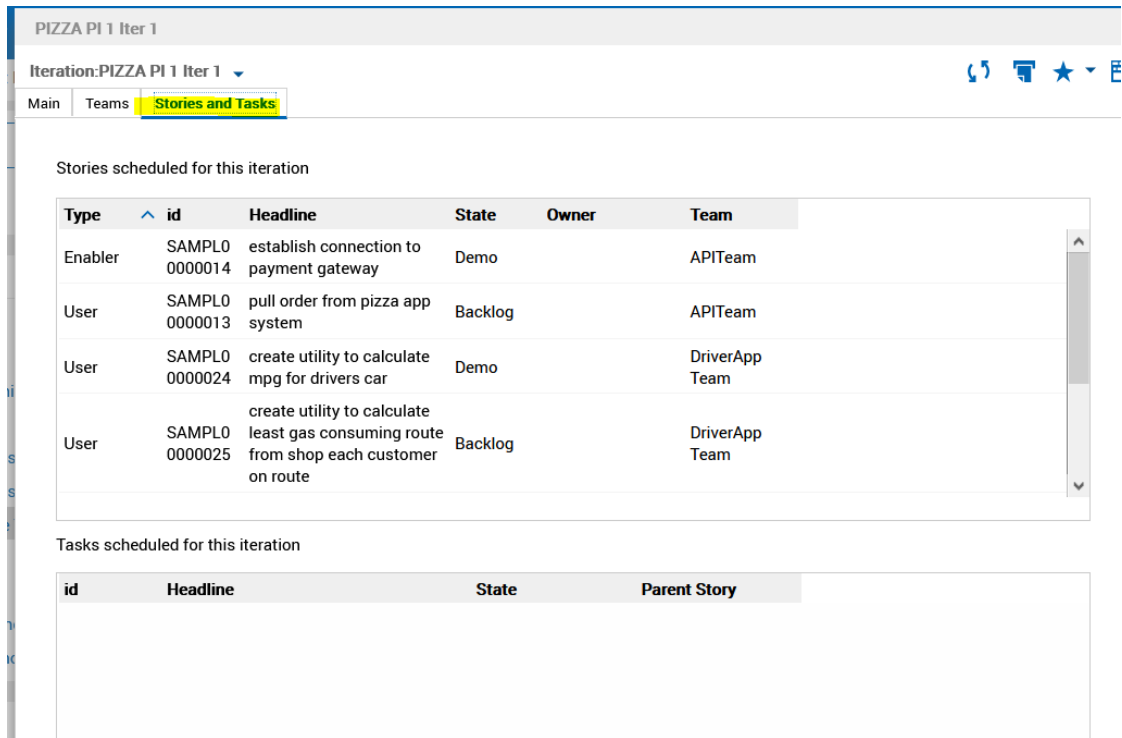


Figure 45 Stories and Tasks Tab on Iteration

8 Work Items

We have gone over all the structural record types to help your team follow SAFe processes. Now we will go over work item records. These records track actual work done by the team. The work items follow the [SAFe Requirements Model](#). For the HCL Compass EssentialSAFe schema, we only track work items associated with Essential SAFe, such as the Feature, Story and Task. We describe these below.

8.1 Features

[Features](#) are work items which are completed within a Program Increment. There are two types of Features – Enablers and Business Features. They have stakeholder interest, meet business objectives, and are completed by being sliced into smaller stories which are completed during iterations within the PI. Once all its stories are complete and tested, the Feature is complete.

The following diagram shows the states and actions available for a Feature. This is the default and suggested flow, but you may skip over states as well. To enter a state, you would run the action pointing into it. For example, to enter the **Implementing** state from any state prior to it, you would use **Implement** action. To enter the **Implementing** state from any state after, you would use the **ReImplement** action.

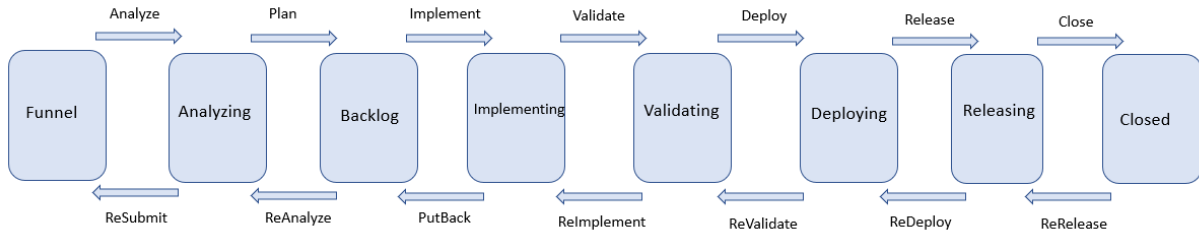


Figure 46 State Flow for Feature

To create a feature, use the **New->Feature** menu option.

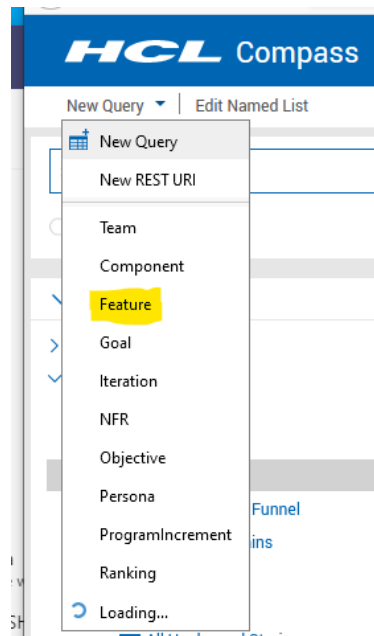


Figure 47 New Feature Menu Option

We will now look at the form for the Feature.

8.1.1 Main

The Main tab has commonly used fields. We describe the fields below. Any field marked with a * in the headline is mandatory.

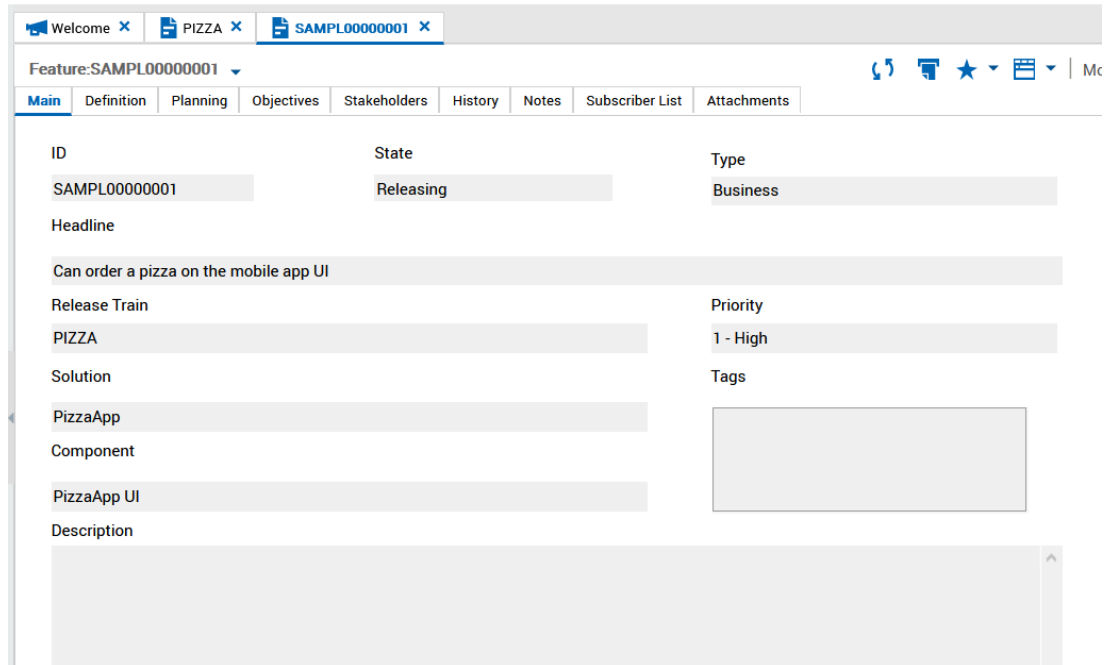


Figure 48 Main Tab on Feature

8.1.1.1 ID

This value is automatically set when you create the record and cannot be changed.

8.1.1.2 State

This value can only be changed using one of the state transition actions shown in Figure 46. On creation, the initial state will be **Funnel**.

8.1.1.3 Type*

A Feature can be either a Business or Enabler Feature. A business feature adds functionality and enablers support the development process and build the [Architectural Runway](#).

8.1.1.4 Headline*

A short description of the Feature.

8.1.1.5 Release Train*

The Feature exists in the context of a Release Train, therefore every Feature must have a Release Train specified.

8.1.1.6 Solution*

A Feature adds value to a solution. Choose the solution to which this Feature applies.

8.1.1.7 Priority*

Choose a priority for this Feature. Remember, you can customize these in the ReleaseTrain Configuration tab (see section 6.9.2).

8.1.1.8 Component

You can optionally choose a component of the solution to be more specific about where the Feature applies.

8.1.1.9 Tags

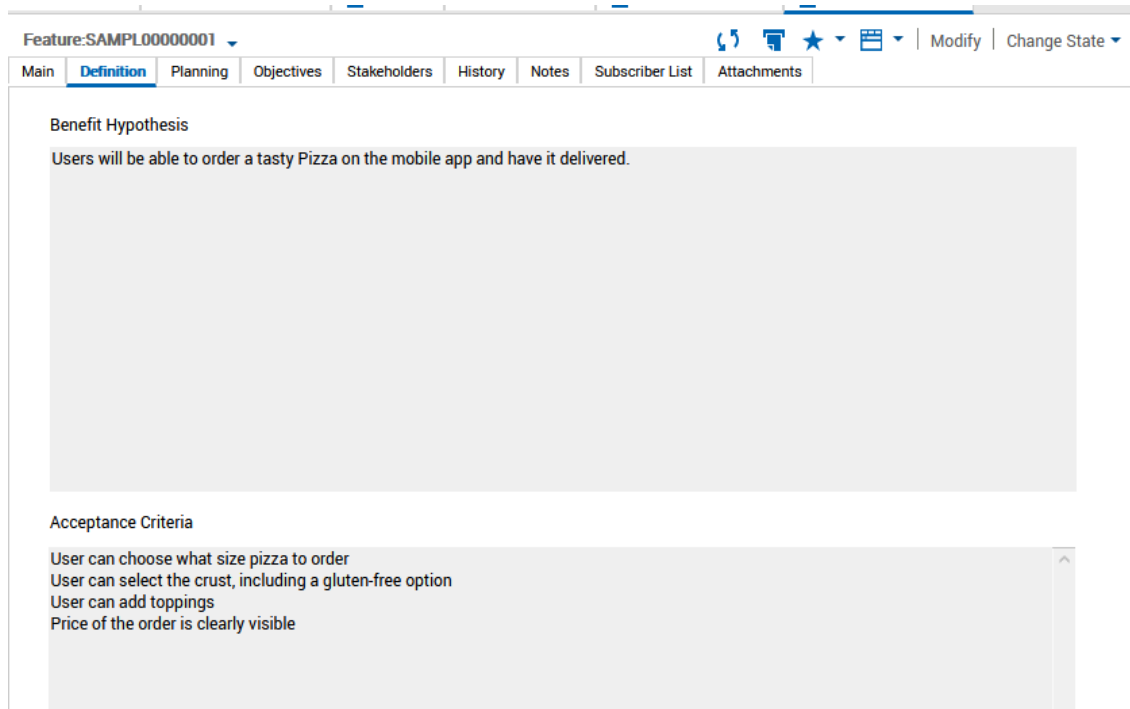
You can add tags to the record for easy searching or categorization.

8.1.1.10 Description

You can put more details here.

8.1.2 Definition

This tab helps you define the Feature. It has just two fields, the Benefit Hypothesis and the Acceptance Criteria. For insight on what to put here, refer to [Features and Capabilities](#) on the Scaled Agile Framework website.



The screenshot shows a software interface for defining a feature. At the top, there is a header bar with the text "Feature:SAMPL0000001" and several icons (refresh, print, star, calendar) followed by "Modify" and "Change State" buttons. Below the header is a navigation menu with tabs: "Main", "Definition" (which is selected and highlighted in blue), "Planning", "Objectives", "Stakeholders", "History", "Notes", "Subscriber List", and "Attachments". The main content area is divided into two sections. The first section is titled "Benefit Hypothesis" and contains a text box with the text "Users will be able to order a tasty Pizza on the mobile app and have it delivered." The second section is titled "Acceptance Criteria" and contains a list of four criteria: "User can choose what size pizza to order", "User can select the crust, including a gluten-free option", "User can add toppings", and "Price of the order is clearly visible".

Figure 49 Definition Tab on Feature

8.1.3 Planning

This tab is used to help plan the Feature.

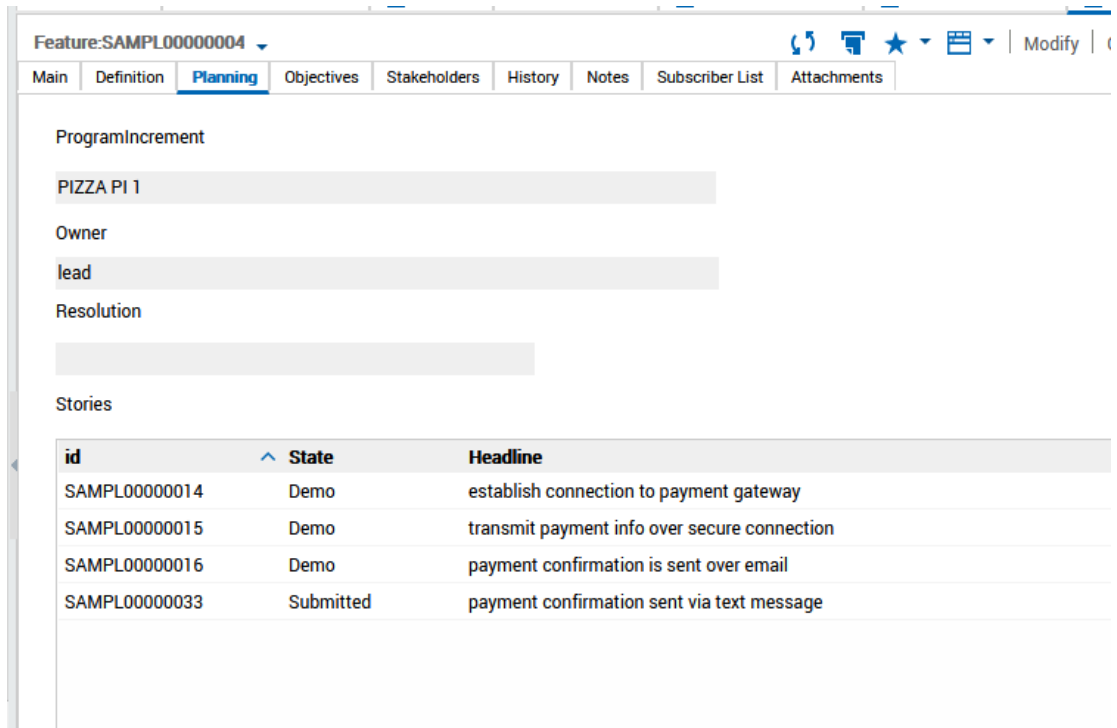


Figure 50 Planning Tab on Feature

8.1.3.1 Program Increment

During [PI Planning](#), the organization leadership decides which features to implement for the coming Program Increment. They can do this by setting the Program Increment field on the Feature. If you have future ProgramIncrement records created, you could also create a [roadmap](#), assigning features to later ProgramIncrements to help you communicate your development horizon.

8.1.3.2 Owner

A business owner or system architect usually creates the Feature and owns it throughout its lifetime.

8.1.3.3 Resolution

when the feature is finally finished, a resolution code will be entered – these codes are customized by the RTE in the Release Train record.

8.1.3.4 Stories

Features get sliced into smaller stories, where the implementation happens. These stories will be listed on this tab. Double-click on any story to view its details.

8.1.4 Objectives

During [PI Planning](#), you typically start with the Features you want to create and then teams will create [Objectives](#) from these. Teams that need to collaborate on a Feature may create separate objective specific to their domain. During the PI Planning, teams can create Objective records directly on the Feature using this tab by clicking the **New** button. Existing objectives can also be added using the **Add** button.

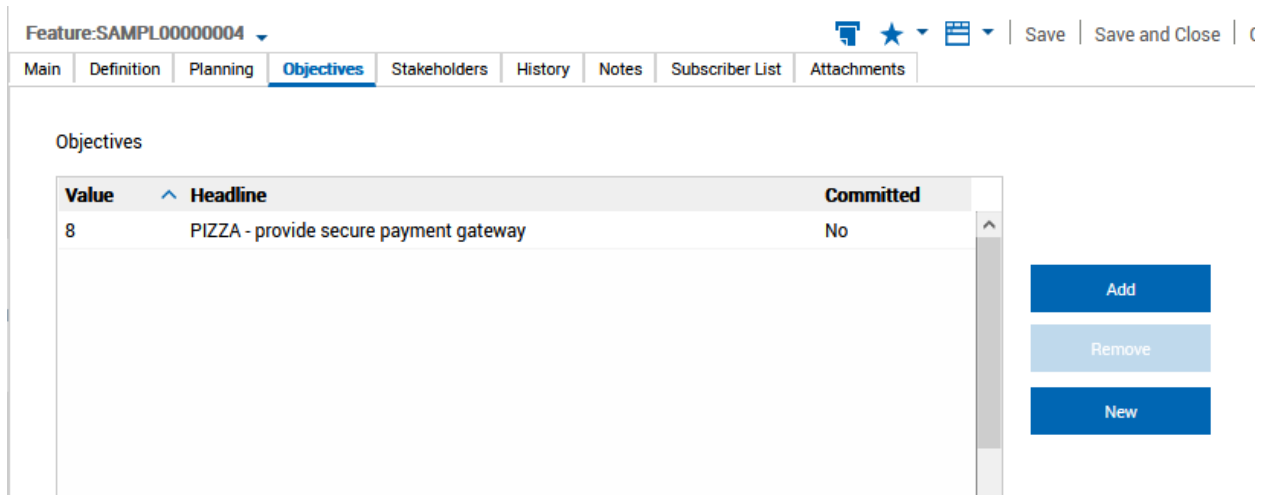


Figure 51 Objectives Tab on Feature

8.1.5 Stakeholders

Stakeholders interested in the enablement of this feature can be logged here – they can be added from existing stakeholder records or created new. These stakeholders can be internal or external (customers)

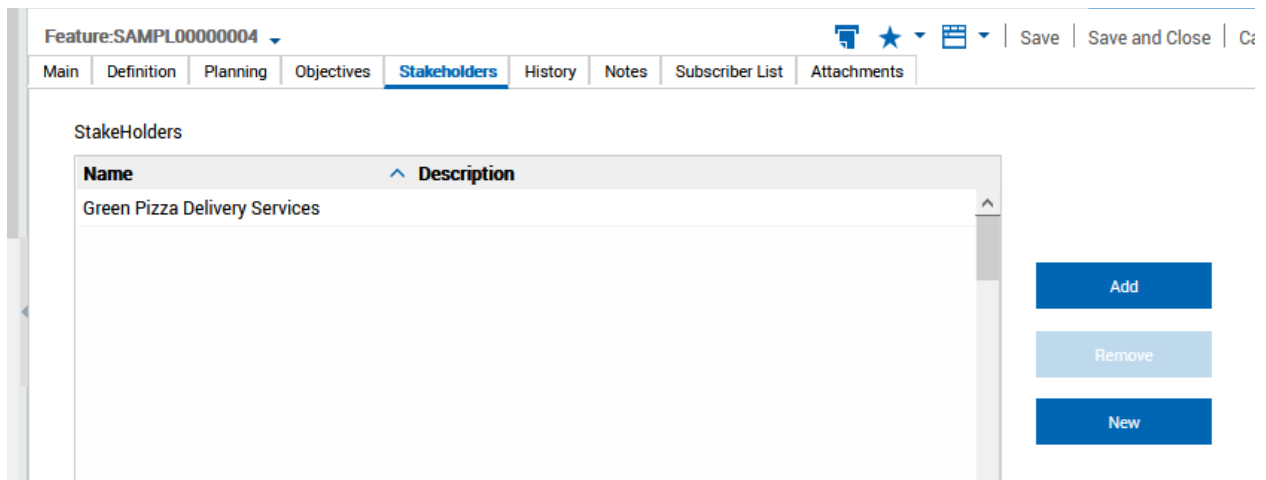


Figure 52 Stakeholders Tab on Feature

8.2 Stories

Stories are work items which are completed within an Iteration. Stories are either sliced from features or may arise from retrospectives or testing (defects). Stories should be sized appropriately so they can fit into a single iteration.

The following diagram shows the states and actions available for a Story. This is the default and suggested flow, but you may skip over states as well. To enter a state, you would run the action pointing into it. For example, to enter the Work state from any state prior to it, you would use **StartWork** action. To enter the Work state from any state after, you would use the **ReWork** action.

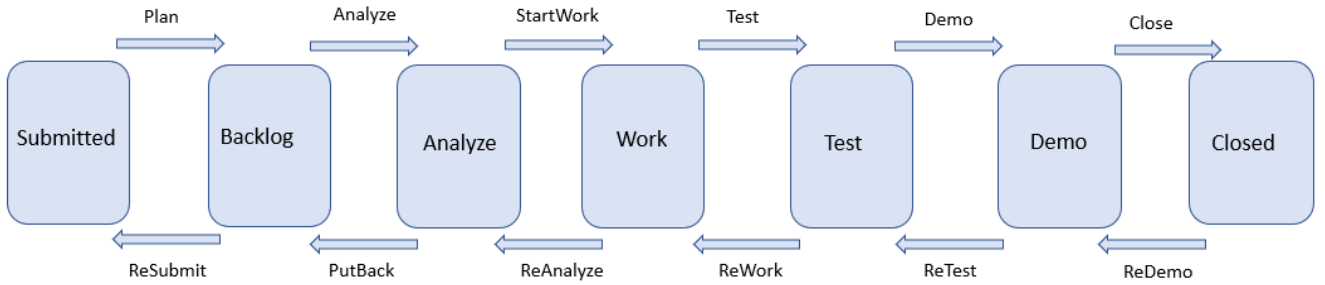


Figure 53 State Flow for Story

There are two ways to create a Story. The easiest way, especially when splitting a Feature into separate Story records, is to run the **Utilities->CreateStory** on the Feature. This will create a new Story and pre-populate Release Train, Solution, Priority, Component, PI and Parent Feature with values from the Feature. This makes slicing a Feature into Story records quick and easy.

The screenshot shows a software interface with a breadcrumb trail: **Feature: SAMPL00000001**. The **Utilities** menu is open, and the **CreateStory** option is highlighted. Below the menu, the feature details are visible:

- ProgramIncrement: PIZZA PI 1
- Owner: [Redacted]
- Resolution: [Redacted]

The **Stories** section contains a table with the following data:

id	State	Headline
SAMPL00000028	Demo	as a customer i want to add multiple topping to a pizza
SAMPL00000029	Demo	as a customer i want to remove topping from a pizza before ordering it
SAMPL00000030	Closed	as a customer i want to order tonning on half of my pizza

Figure 54 Running CreateStory Action on Feature

Feature: SAMPL0000001

Main Definition Planning Objectives Stakeholders History Notes Subscriber List Attachments

ID *Story

SAMPL00000046 Story: SAMPL00000046 Save Save and Close

Headline Main Definition Planning Goals History Notes Subscriber List Attachments

Can order a PizzaApp

Release Train ID State Type Persona

PIZZA SAMPL00000046 Submitted User

Solution Headline

PizzaApp This is my new story!

Component Release Train Priority

PizzaApp UI PIZZA 1 - High

Description Solution Tags

PizzaApp

Component PizzaApp UI

Template: Load

Figure 55 New Story with Prepopulated Fields from Feature

Some stories are not split from a Feature. These stories can also be created from the **New->Story** menu option.

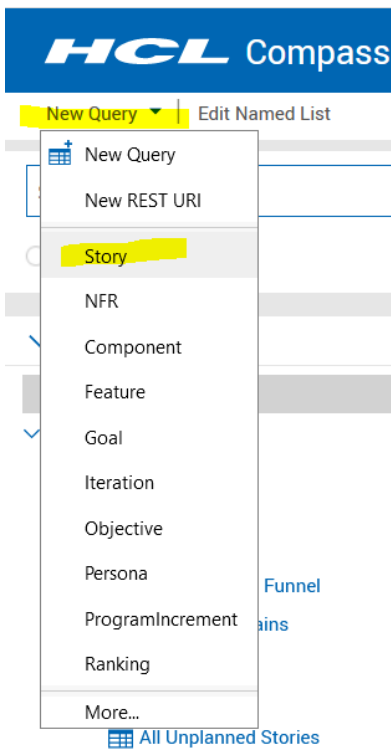


Figure 56 Creating Story with New->Story

We will now take a closer look at the Story form.

8.2.1 Main

The Main tab has commonly used fields. We describe the fields below. Any field marked with a * in the headline is mandatory.

The screenshot shows the 'Main' tab for a Story with ID SAMPL00000037. The form includes the following fields:

- ID:** SAMPL00000037
- State:** Submitted
- Type:** User
- Persona:** (empty dropdown)
- * Headline:** (empty text field)
- * Release Train:** (empty dropdown)
- * Priority:** (empty dropdown)
- * Solution:** (empty dropdown)
- Component:** (empty dropdown)
- Description:** (empty text area)
- Tags:** (empty text area with a plus icon)

Figure 57 Main Tab on Story

8.2.1.1 ID

This value is automatically set when you create the record and cannot be changed.

8.2.1.2 State

This value can only be changed using one of the state transition actions shown in Figure 53. On creation, the initial state will be **Submitted**.

8.2.1.3 Type*

Scaled Agile Framework defines two types of stories, [User](#) and [Enabler](#) stories. HCL Compass EssentialSAFe schema also defines a third type of Story, the Defect. This can be used to represent defects in the product that need to be fixed and can fit in an iteration.

8.2.1.4 Headline*

A short description of the Story.

8.2.1.5 Release Train*

The Story exists in the context of a Release Train, therefore every Story must have a Release Train specified.

8.2.1.6 Solution*

A Story adds value to a solution. Choose the solution to which this Story applies.

8.2.1.7 Priority*

Choose a priority for this Story. Remember, you can customize these in the ReleaseTrain Configuration tab (see section 6.9.2).

8.2.1.8 Component

You can optionally choose a component of the solution to be more specific about where the Story applies.

8.2.1.9 Tags

You can add tags to the record for easy searching or categorization.

8.2.1.10 Description

You can put more details here.

8.2.1.11 Persona

If you use [design thinking](#), which can help create desirable products, you can choose a Persona that this story applies to.

8.2.2 Definition

This tab helps you define the Story. It has just one field, the Acceptance Criteria. For insight on what to put here, refer to [Story](#) page on the Scaled Agile Framework website. You might describe a series of tests using [Behavior Driven Development](#) practices.

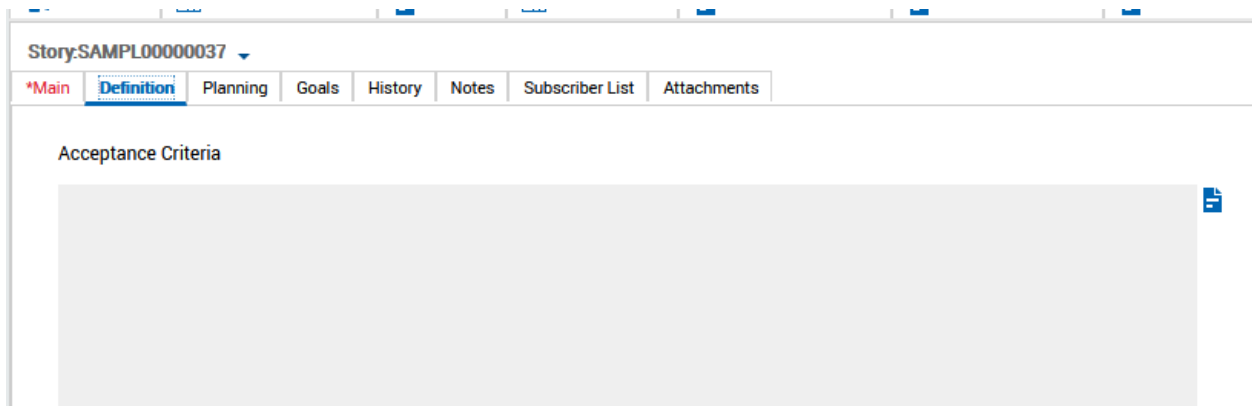


Figure 58 Definition Tab on Story

8.2.3 Planning

This tab can be used to help plan the Story.

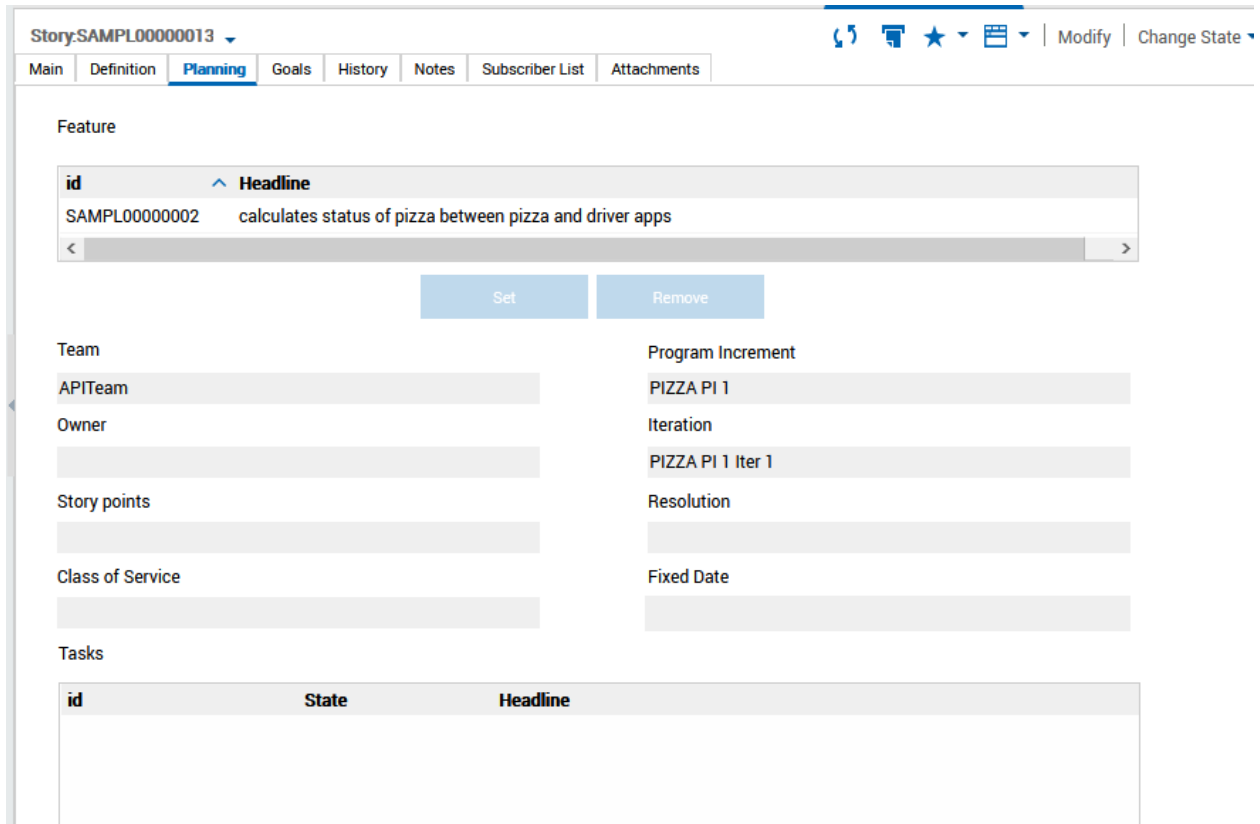


Figure 59 Planning Tab on Story

8.2.3.1 Feature

A story can have a parent Feature. If the story was not created from a feature directly, you can add one here or if there is no parent (as may be the case for a defect or some enablers), leave the field blank.

Typically, a story is created as a child of a Feature, sliced out as a part of a PI Planning exercise. That is not always the case. A story is not required to have a parent. For example, a defect story found during development or testing might not have a parent Feature. An enabler story that was created internally by the team, such as during retrospective, might not have a parent Feature.

8.2.3.2 Team

The team that will work on the story.

8.2.3.3 Program Increment

This field is the ProgramIncrement this story is planned for. This list will be limited by whatever ReleaseTrain the record is assigned to.

8.2.3.4 Owner

The person who will do the work. This list will not be limited in any way.

8.2.3.5 Iteration

The Sprint in which the story will be completed. This list will be limited by the selected Program Increment.

8.2.3.6 Story Points

This field contains the estimate for the relative work effort for a [story](#). The list is limited to whatever story points are configured on the ReleaseTrain.

8.2.3.7 Resolution

When the work item is complete, a resolution code can be assigned here. The list is limited to whatever resolution codes are configured on the ReleaseTrain.

8.2.3.8 Class of Service

The Class of Service is a way to optimize execution of work in [Kanban](#). The list is limited to whatever classes of service are configured on the ReleaseTrain. The “0 – Fixed Date” class of service is built-in and always first in the list. If this is chosen, then **Fixed Date** becomes a mandatory field.

8.2.3.9 Fixed Date

If the **Class of Service** is set “0 – Fixed Date”, you must set a **Fixed Date** to indicate when this story must be done.

8.2.4 Tasks

In SAFe, [Tasks](#) are short work items (usually in the measurement of hours) that can be used to further break down stories and to help teams understand how much work is involved. Tasks are optional and used mainly by newer teams until they can plan and work entirely at the Story level. We learn more about Task records in section 8.3.

8.2.5 Goals

During [Iteration Planning](#), teams create [Goals](#), as described in section 7.2.1.1.2 to help align with the teams Objectives (see section 7.1.3). To demonstrate that the stories they choose are aligned with these goals, you can add these goals here. Conversely, you could create new goals on the stories, and add them to the TeamIteration Goals tab during iteration planning.

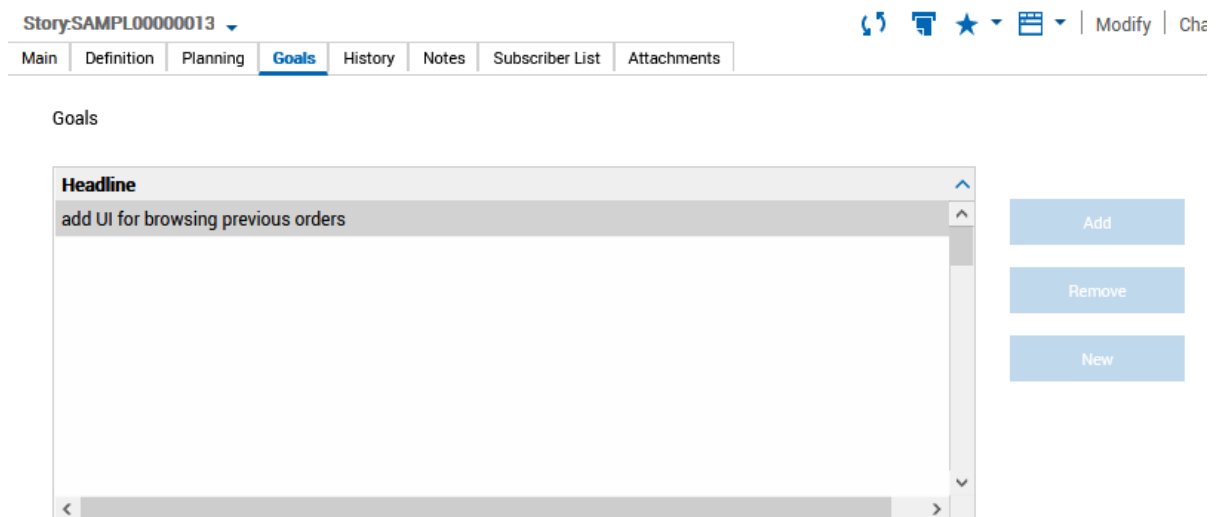


Figure 60 Goals Tab on Story

8.3 Tasks

Tasks were introduced in section 8.2.4. The Task is the smallest work item in EssentialSAFe. The Task also does not have a type nor a Story Point estimate. The effort in a Task is so short it is smaller than one story point (usually modelled as one person-day). [Tasking stories](#) is an optional process in SAFe. It is often used by newer teams to help them better size stories. More experienced teams may forego creation of tasks and stick with stories as the smallest work item. In HCL Compass EssentialSAFe, a Task record does not need to have a parent Story, but it usually does.

The state flow for a Task is shown below, and it can go forward or backward like the other work items.

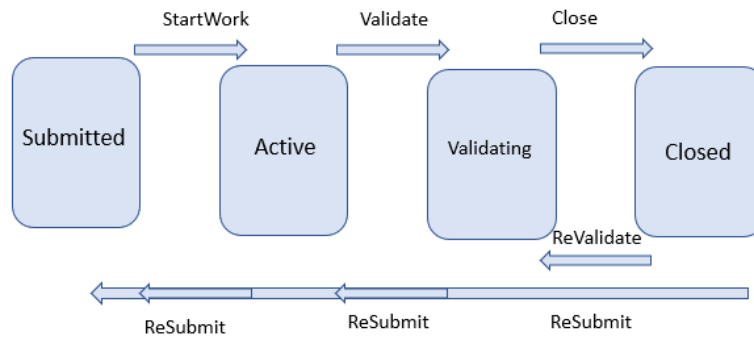


Figure 61 State Flow for Task

There are two ways to create a Task. The easiest way, especially when splitting a Story into separate Task records, is to run the **Utilities->CreateTask** on the Story. This will create a new Task and pre-populate Release Train, Solution, Priority, Component, PI and Parent Story with values from the Story. This makes slicing a Story into Task records quick and easy.

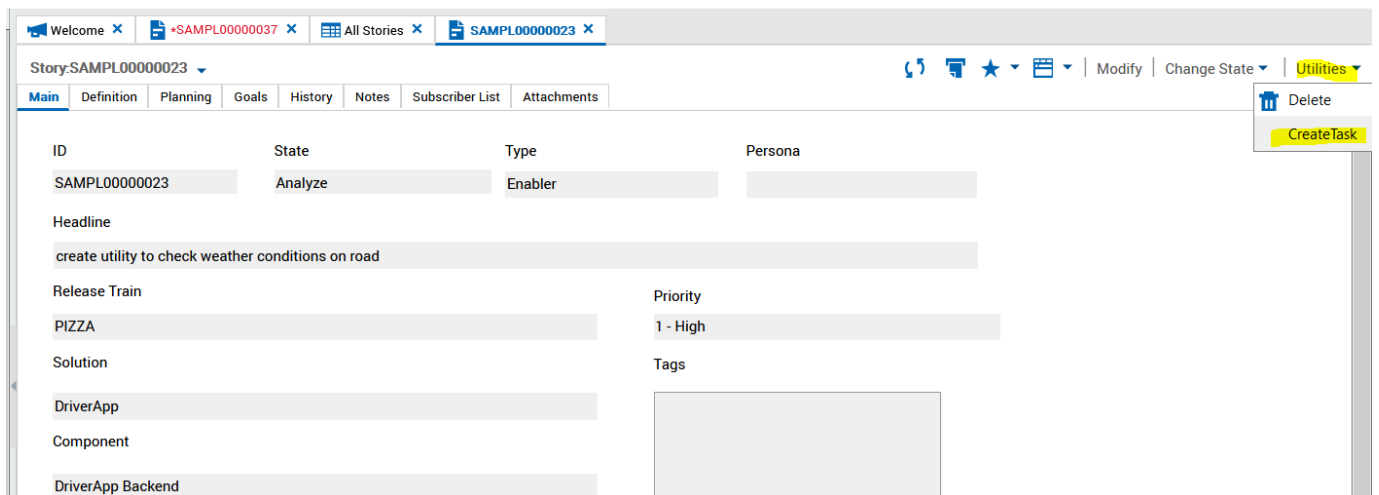


Figure 62 CreateTask Action on Story

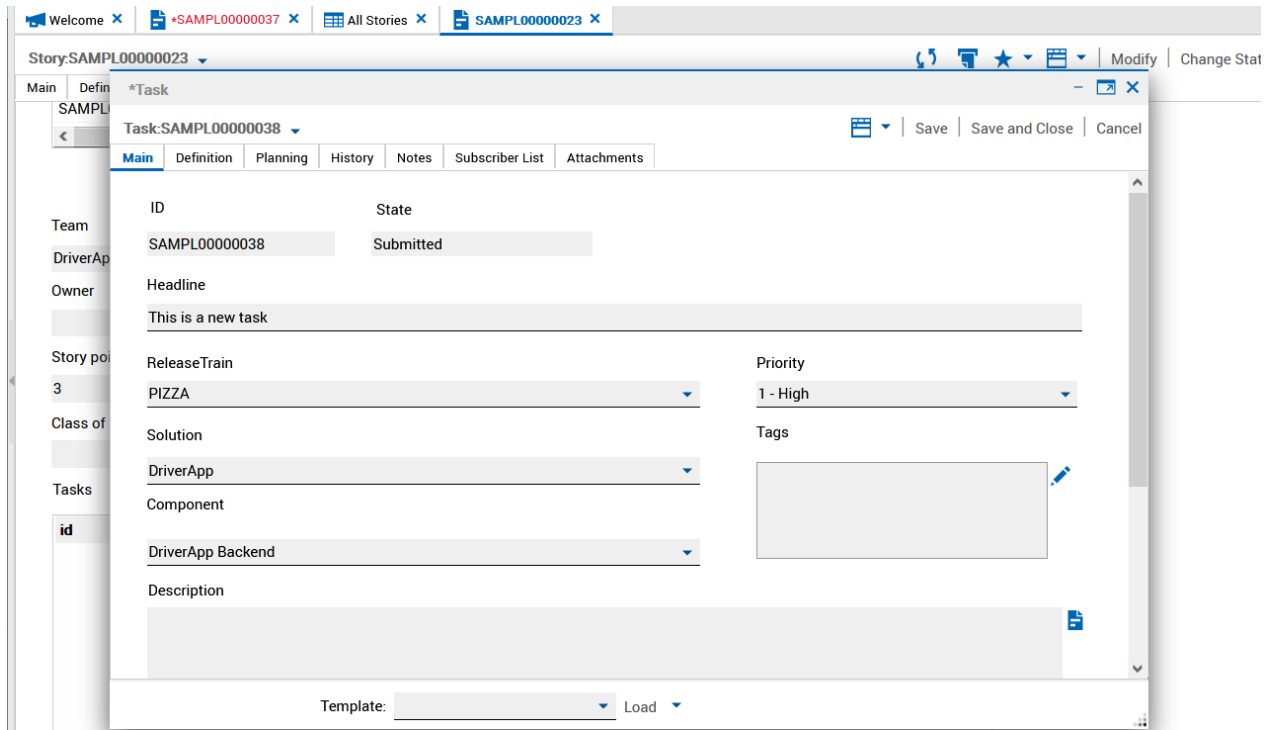


Figure 63 New Task with Prepopulated Fields from Story

Some Tasks are not split from a Story. These Tasks can also be created from the **New->Task** menu option.

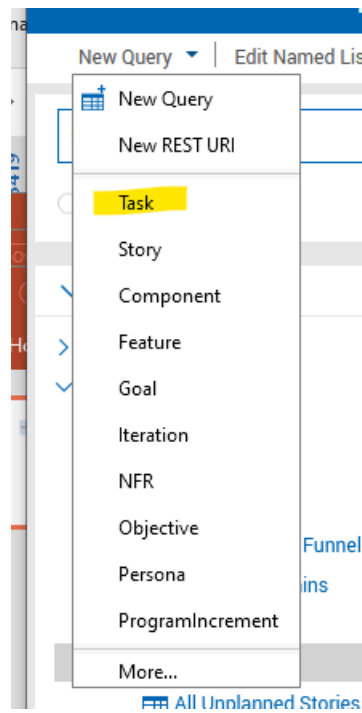


Figure 64 Creating Task with New->Task

8.3.1 Main

The Main tab has commonly used fields. We describe the fields below. Any field marked with a * in the headline is mandatory.

The screenshot shows a web interface for a task record. At the top, there is a breadcrumb 'Task:SAMPL00000038' and navigation buttons for 'Save', 'Save and Close', and 'C'. Below this is a tabbed interface with 'Main' selected, along with other tabs: 'Definition', 'Planning', 'History', 'Notes', 'Subscriber List', and 'Attachments'. The main form area contains the following fields:

- ID:** A text field containing 'SAMPL00000038'.
- State:** A text field containing 'Submitted'.
- * Headline:** A large text area for a short description.
- * ReleaseTrain:** A dropdown menu.
- * Priority:** A dropdown menu.
- * Solution:** A dropdown menu.
- Component:** A dropdown menu.
- Tags:** A text area with a blue pencil icon for editing.
- Description:** A large text area with a blue document icon for adding attachments.

Figure 65 Main Tab on Task

8.3.1.1 ID

This value is automatically set when you create the record and cannot be changed.

8.3.1.2 State

This value can only be changed using one of the state transition actions shown in Figure 61. On creation, the initial state will be **Submitted**.

8.3.1.3 Headline*

A short description of the Task.

8.3.1.4 Release Train*

The Task exists in the context of a ReleaseTrain, therefore every Task must have a Release Train specified.

8.3.1.5 Solution*

A Task add value to a solution. Choose the solution to which this Task applies.

8.3.1.6 Priority*

Choose a priority for this Task. Remember, you can customize these in the ReleaseTrain Configuration tab (see section 6.9.2).

8.3.1.7 Component

You can optionally choose a component of the solution to be more specific about where the Task applies.

8.3.1.8 Tags

You can add tags to the record for easy searching or categorization.

8.3.1.9 Description

You can put more details here.

8.3.2 Definition

This tab helps you define the Task. It has just one field, the Acceptance Criteria. For insight on what to put here, refer to [Story](#) page on the Scaled Agile Framework website. You might describe a series of tests using [Behavior Driven Development](#) practices. Tasks are so small that you might forego defining the acceptance criteria on the Task and instead define them all on the Story.

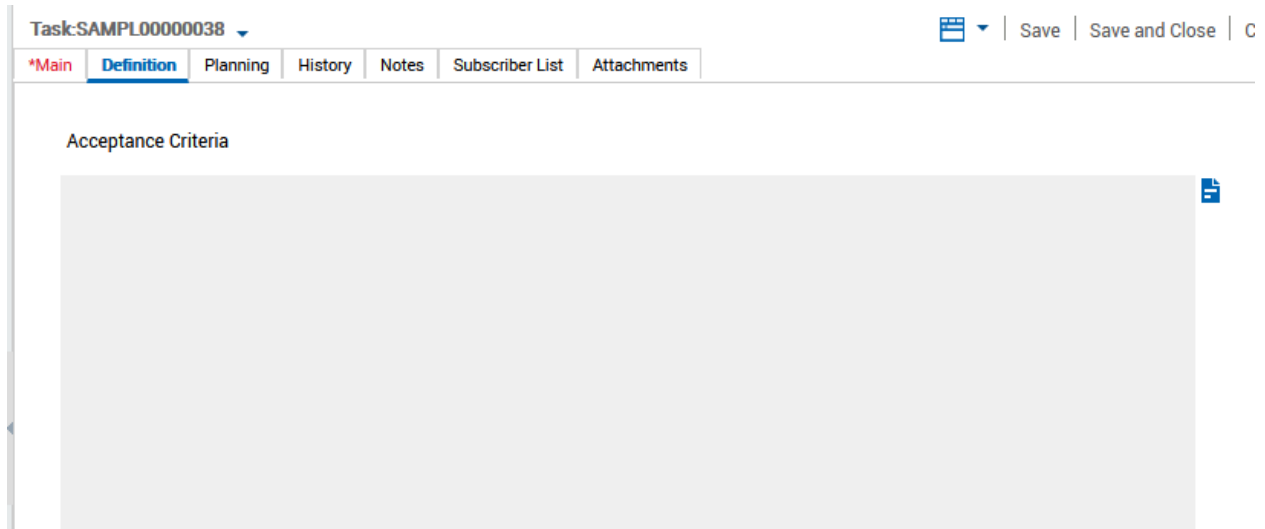


Figure 66 Definition Tab on Task

8.3.3 Planning

This tab can be used to help plan the Task. Tasks assigned to an Iteration will appear on the **Stories and Tasks** tab on the Iteration. It should be noted that in HCL Compass EssentialSAFe, tasks will not show up in the team Kanban or backlog queries along with stories. You can create separate queries to track Tasks.

Task: SAMPL00000038

Save | Save and Close | Cancel

*Main | Definition | **Planning** | History | Notes | Subscriber List | Attachments

Story

id	Headline	Iteration

Set Remove

Team

Program Increment

Owner

Iteration

Resolution

Figure 67 Planning Tab on Task

8.3.3.1 Story

A Task can have a parent Story. If the Task was not created from a Story directly, you can add one here or if there is no parent (as may be the case for a defect or some enablers), leave the field blank.

Typically, a task is created as a child of a story, sliced out as a part of [the Iteration Planning](#) exercise. That is not always the case. A Task is not required to have a parent. For example, a task may have been created internally by the team during retrospective.

8.3.3.2 Team

The team that will work on the task.

8.3.3.3 Program Increment

This field is the ProgramIncrement this task is planned for. This list will be limited by whatever ReleaseTrain the record is assigned to.

8.3.3.4 Owner

The person who will actually do the work. This list will not be limited in any way.

8.3.3.5 Iteration

The Sprint in which the story will be completed. This list will be limited by the selected Program Increment.

8.3.3.6 Resolution

When the work item is complete, a resolution code can be assigned here. The list is limited to whatever resolution codes are configured on the ReleaseTrain.

8.4 Common Work Item Tabs

If you are using the out of the box HCL Compass Essential SAFe schema, you will find that each of the work items described above have a set of common tabs. If, on the other hand, you are applying the EssentialSAFe package to an existing schema, some of these tabs will not be present. You can add them by enabling the work items with the respective package they come from.

8.4.1 History

These fields are read-only and track who submitted, closed, and last modified the record along with a date and time stamp. There is also a standard history control showing the entire work item history. This tab is included with the EssentialSAFe package.

8.4.2 Notes

The notes tab is a place to keep ongoing notes about the progress of the work item and communicate any questions with the team or stakeholders. The Notes tab can be added to existing schemas using the [Notes](#) package.

8.4.3 Subscriber List

The Subscribers List tab comes from the [EmailPlus](#) package. The EmailPlus package has enhanced email notification capabilities. We recommend using this email notification package for the best user experience. A Compass admin must enable and configure EmailPlus before it starts sending notifications. Refer to the [EmailPlus documentation](#) for how to do this.

8.4.4 Attachments

Attachments is a place to store files related to the record. This tab is added by the [Attachments](#) package.

Figure 68 dddd

9 Queries

HCL Compass EssentialSAFe ships with a number of generic queries out-of-the-box, and it is also possible to generate more specific queries running the **CreateBacklogsAndKanbans** action on the ReleaseTrain. The following sections describe these queries and suggest how you can use them in SAFe workflow.

9.1 Public Queries

The EssentialSAFe package includes some queries to make it easy to find things. These queries are generic in scope and return many more records than you may initially need, but they can be helpful to someone just getting started to find their data. The queries can be modified to add display columns and filters. You can then save the modified (and therefore more useful) queries to your personal or public workspace.

A few common use cases for these queries are described below.

9.1.1 Viewing Scrum Notes

A common use case is looking at your team's scrum notes. We describe two methods for finding your team's scrum notes, by running the **All Release Trains** query.

9.1.1.1 Using Forms.

Run the “All Release Trains” query and find your release train in the result set. Click on it to open its form.

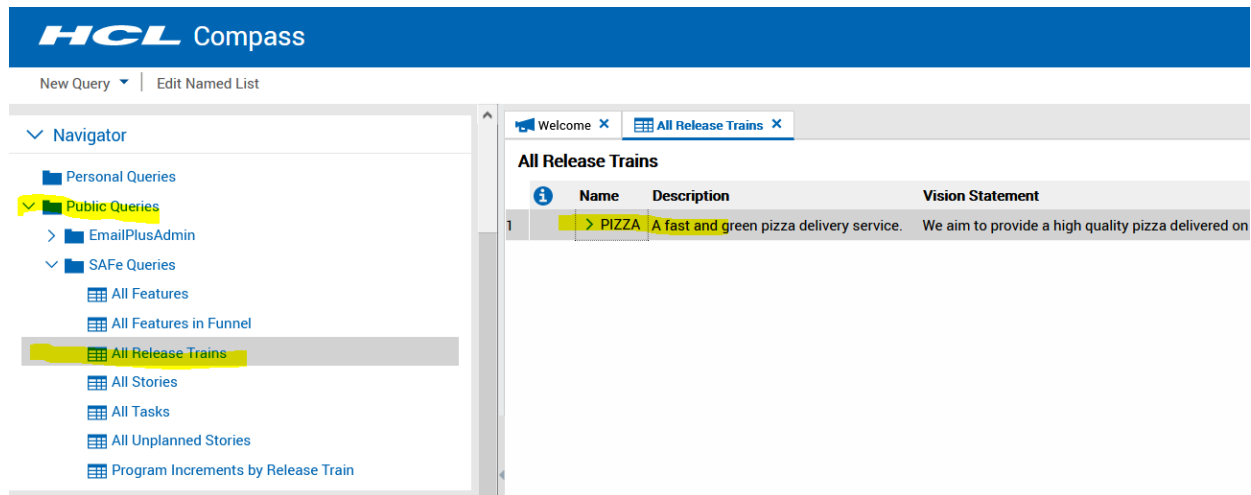


Figure 69 All Release Trains Query

On the Program Increments tab, find the one you want the scrum notes for and open it (e.g. double click).

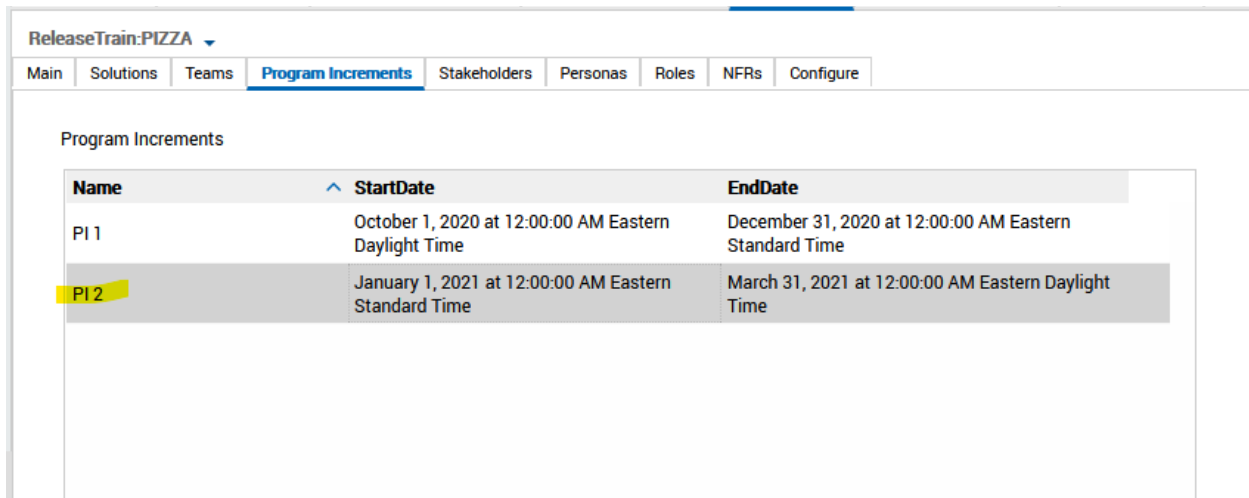


Figure 70 Choose PI from ReleaseTrain

Find your Team in the Teams tab and double click on it.

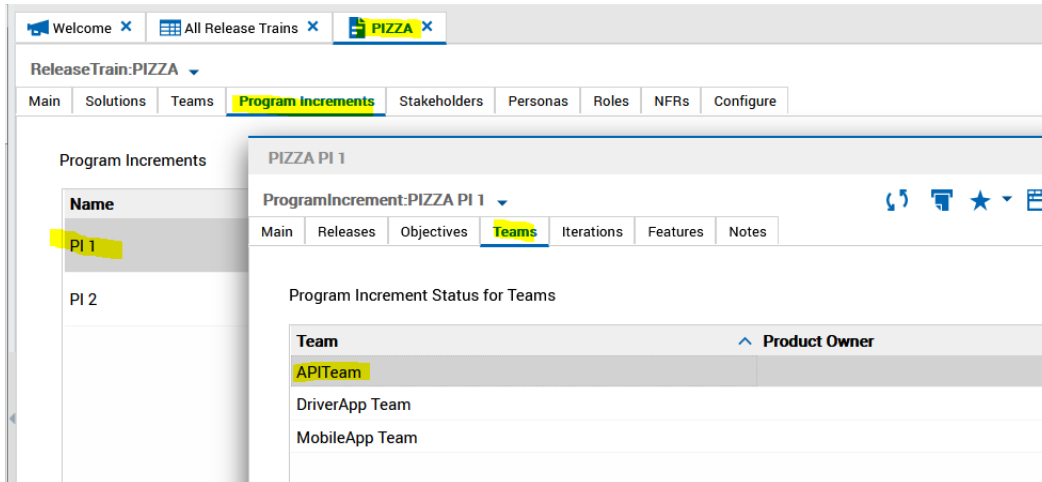


Figure 71 Choose Team from PI

On the TeamPI form that opens, switch to the Notes tab. Here are the daily scrum notes your team.

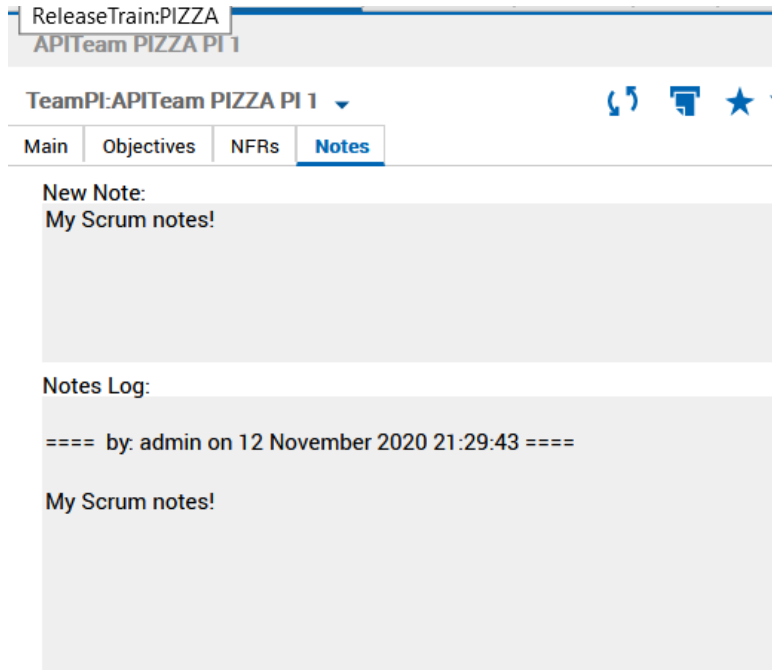


Figure 72 Scrum Notes on TeamPI

To add more scrum notes, run the **Modify** action on the TeamPI and add a new note. Only the product owner for the team can do this.

9.1.1.2 Using Tree View

The second method of finding the same information uses the **Tree View** feature in the resultset. Run the same query and expand the twisty controls, as in the figure below, to get to right TeamPI.

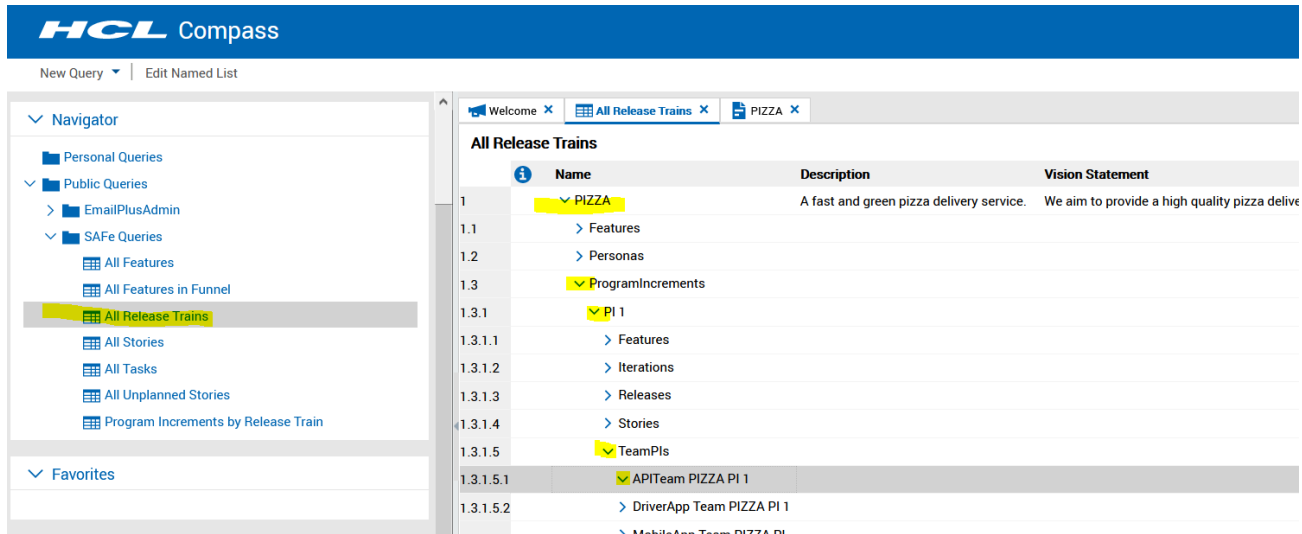


Figure 73 Choose Correct TeamPI from Tree View

Click on the TeamPI corresponding to your team and the same form as in Figure 72 appears.

9.1.2 Triage Work Items

Another common use case is to perform triage on new work items such as new features and stories. [New features](#) begin in the Funnel state (see Figure 46). Business owners and architects may meet to assign new features to subject matter experts for analysis. To find features that are in the funnel state you can run the **All Features in Funnel**.

Similarly, new stories will begin in the submitted state. The **Plan** action takes it to the **Backlog** state. If the story was split from a Feature or created as part of a retrospective, it may already have been placed in the backlog during PI planning. In some cases, you may create a story that has not been planned yet. You can find them with the **All Unplanned Stories** query.

The following queries are included with the tool

9.1.2.1 All Features

This is a query that lists all features in the database.

9.1.2.2 All Features in Funnel

This is a query that lists all features in the **Funnel** state in the database.

9.1.2.3 All Release Trains

This is a query that lists all ReleaseTrain records in the database.

9.1.2.4 All Stories

This is a query that lists all stories in the database.

9.1.2.5 All Tasks

This is a query that lists all tasks in the database.

9.1.2.6 All Unplanned Stories

This is a query that lists all stories in the **Submitted** state in the database.

9.1.2.7 Program Increments by Release Train

This is a dynamic query. When you run it, it will ask you to choose a ReleaseTrain. The query returns all of the ProgramIncrement records within that ReleaseTrain. This can be helpful in planning. If no ReleaseTrain is chosen, all ProgramIncrement records for every ReleaseTrain will be returned.

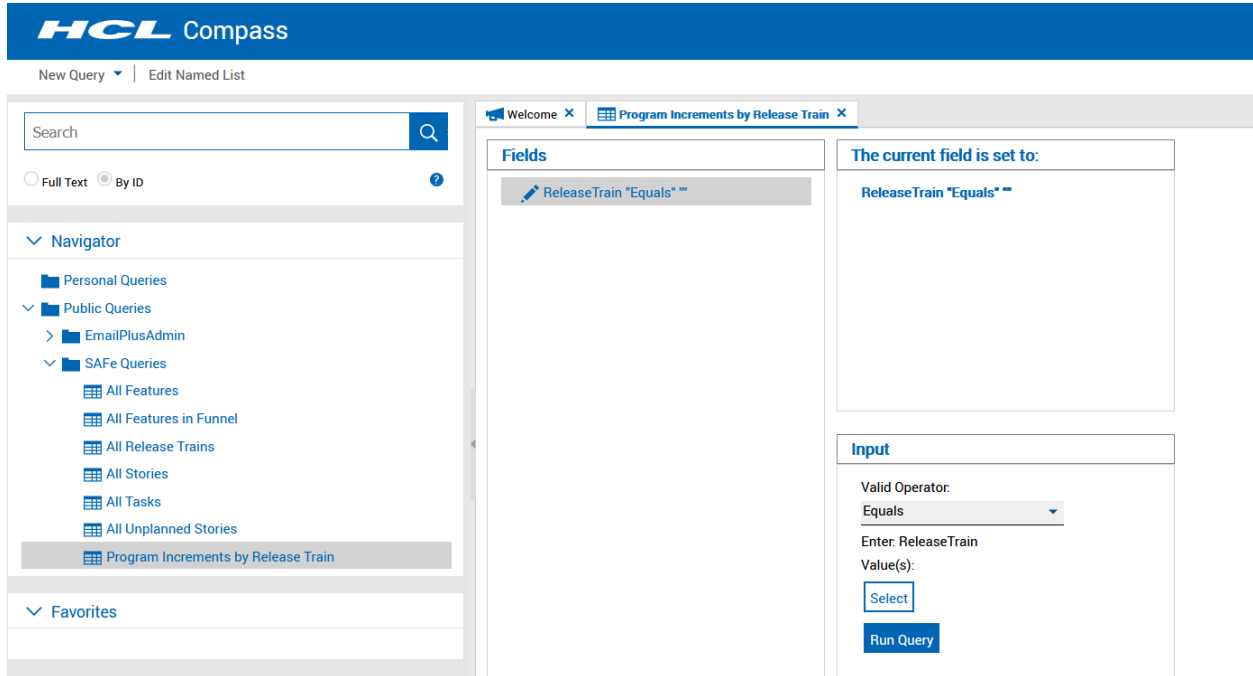


Figure 74 Running Program Increments by Release Train

9.2 Personal Queries

HCL Compass EssentialSAFE allows users to create backlog and Kanban queries in their own personal workspace. This can save you the time and hassle of creating the queries yourself, which can be tedious. To create these queries, open the ReleaseTrain record which you want to create the queries for. You can use the **All Release Trains** public query, or you can just type in the name of the ReleaseTrain in the search box. After opening the Release train run the **Utilities->CreateBacklogsAndKanbans** action.

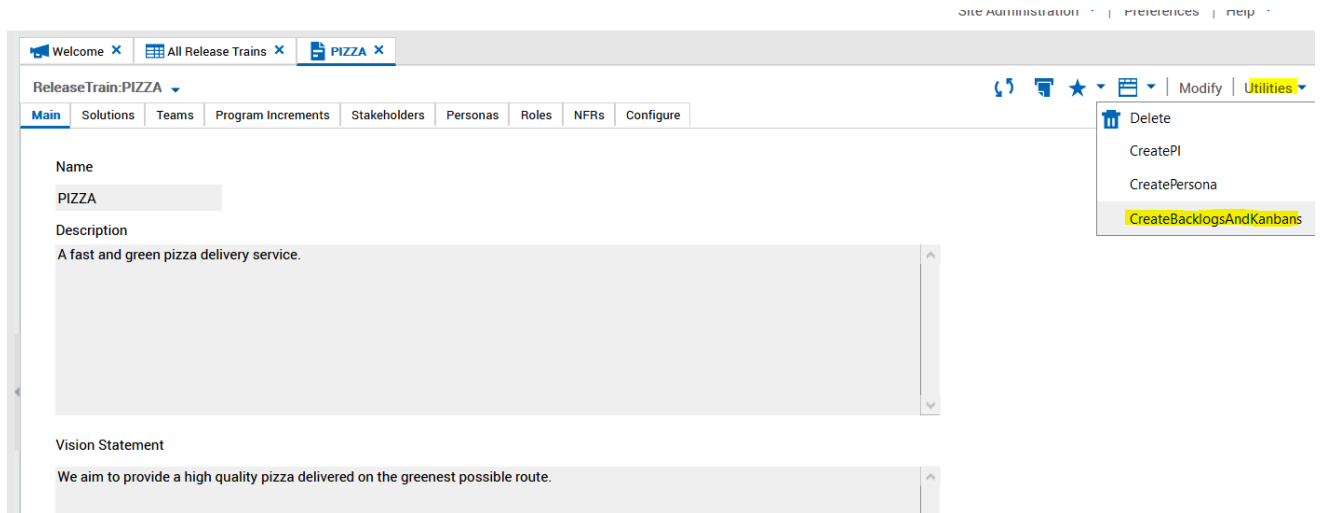


Figure 75 Running CreateBacklogsAndKanbans Action

In your Personal Queries workspace, there will now exist a folder “SAFE Queries for <rt>” where <rt> is the name of your Release Train.

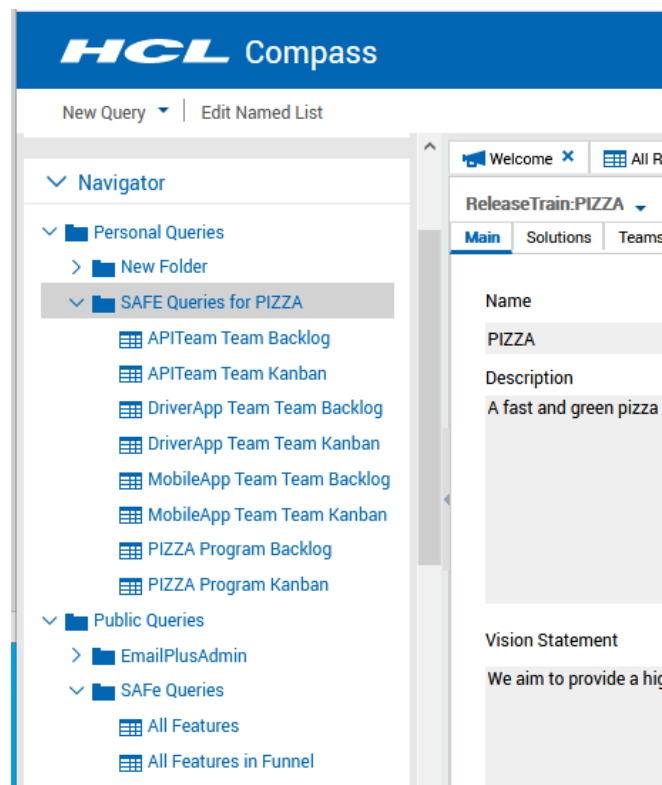


Figure 76 New Folder Created in Personal Queries

In that folder, you’ll see a series of queries – a backlog and Kanban pair for every team on the ReleaseTrain and a Program level backlog and Kanban named after the release train. If the queries don’t show up after running the action, refresh the workspace or logout and back in again to refresh the workspace.

9.2.1 Kanban

This query shows all the records of a given type, regardless of the states, and timebox. In the case of a “Program Kanban” it is for all the features of the Release Train. In the case of the “Team Kanban” it is for all the stories for a specific team in the Release Train.

The Program Kanban is first sorted by State (starting with Funnel), then by Priority within the States.

PIZZA Program Kanban							Previous	1 - 12	of 12	Next
	id	Type	State	Priority	Headline	Solution				
1	> SAMPL00000010	Business	Funnel	1 - High	generate delivery list for driver app based on environmental impact	PizzaApp				
2	> SAMPL00000012	Business	Funnel	2 - Medium	customer and driver can communicate with each other without exchanging phone numbers	DriverApp				
3	> SAMPL00000011	Business	Funnel	3 - Low	show user facts about environment while waiting for pizza	PizzaDash				
4	> SAMPL00000007	Business	Analyzing	3 - Low	ability to call pizza shop back about order from dashboard	PizzaDash				
5	> SAMPL00000002	Business	Backlog	1 - High	calculates status of pizza between pizza and driver apps	PizzaDash				
6	> SAMPL00000005	Business	Backlog	1 - High	conveyor belt shows pizza being made, baked, boxed, and driven	PizzaDash				
7	> SAMPL00000003	Business	Backlog	2 - Medium	driver alerted to pick up delivery through driver app	DriverApp				
8	> SAMPL00000006	Business	Backlog	3 - Low	customer saves frequent order info into profile	PizzaApp				
9	> SAMPL00000009	Business	Backlog	3 - Low	recalculates order of deliveries based of environmental condition changes	DriverApp				
10	> SAMPL00000008	Business	Implementing	1 - High	driver given greenest possible route	DriverApp				
11	> SAMPL00000001	Business	Releasing	1 - High	Can order a pizza on the mobile app UI	PizzaApp				
12	> SAMPL00000004	Business	Releasing	1 - High	process credit card payment	PizzaApp				

Figure 77 Program Kanban Result Set

The Team Kanban is a result set sorted by State, Class of Service, and Priority.

APITeam Team Kanban											Previous	1 - 10	of 10	Next			
	id	Type	State	Priority	Headline	Owner	ClassOfService	FixedDate	Solution	StoryPoints							
1	> SAMPL00000027	Enabler	Submitted	1 - High	send time to dash from drive app when driver gets pizza				PizzaDash								
2	> SAMPL00000032	Enabler	Submitted	2 - Medium	need a way for cooking log data to get to dash				PizzaDash								
3	> SAMPL00000036	User	Submitted	3 - Low	random fact generator				PizzaDash								
4	> SAMPL00000013	User	Backlog	1 - High	pull order from pizza app system				PizzaDash								
5	> SAMPL00000020	Enabler	Backlog	3 - Low	design windows for facts				PizzaDash								
6	> SAMPL00000021	User	Backlog	3 - Low	as a user i would like the app to remember my favorite pizza				PizzaApp	3							
7	> SAMPL00000026	User	Backlog	3 - Low	as a user i need to allow app to have access to phone dialer				PizzaDash	8							
8	> SAMPL00000014	Enabler	Demo	1 - High	establish connection to payment gateway				PizzaApp	5							
9	> SAMPL00000015	Enabler	Demo	1 - High	transmit payment info over secure connection				PizzaApp	5							
10	> SAMPL00000016	User	Demo	3 - Low	payment confirmation is sent over email				PizzaApp	3							

Figure 78 Team Kanban Result Set

Inline editing can be used to update work items directly in the Kanban using **Right Click->Inline Edit**.

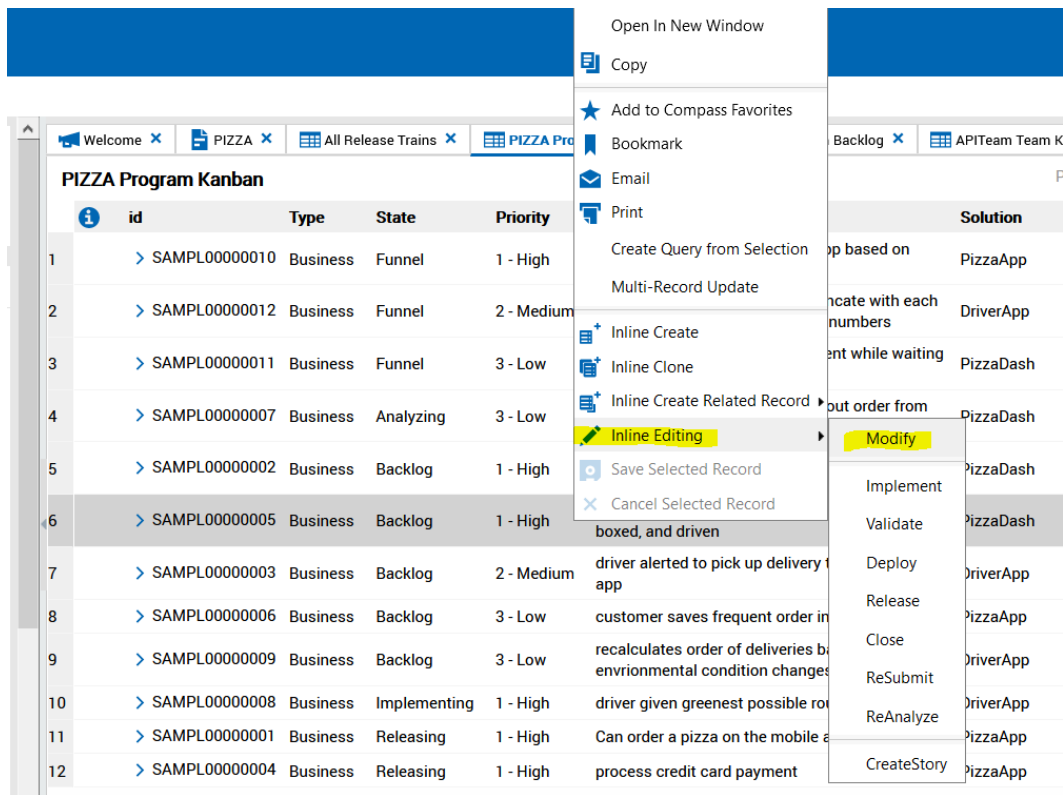


Figure 79 Inline-Edit in Kanban

PIZZA Program Kanban					Inline Mode Action: Modify	
	id	Type	State	Priority	Headline	Solution
1	> SAMPL00000010	Business	Funnel	1 - High	generate delivery list for driver app based on environmental impact	PizzaApp
2	> SAMPL00000012	Business	Funnel	2 - Medium	customer and driver can communicate with each other without exchanging phone numbers	DriverApp
3	> SAMPL00000011	Business	Funnel	3 - Low	show user facts about environment while waiting for pizza	PizzaDash
4	> SAMPL00000007	Business	Analyzing	3 - Low	ability to call pizza shop back about order from dashboard	PizzaDash
5	> SAMPL00000002	Business	Backlog	1 - High	calculates status of pizza between pizza and driver apps	PizzaDash
6	> SAMPL00000005	Business	Backlog	3 - Low	shows pizza being made, baked, boxed, and driven	PizzaDash
7	> SAMPL00000003	Business	Backlog	2 - Medium	driver alerted to pick up delivery through driver app	DriverApp
8	> SAMPL00000006	Business	Backlog	1 - High	customer saves frequent order info into profile	PizzaApp
9	> SAMPL00000009	Business	Backlog	3 - Low	recalculates order of deliveries based of environmental condition changes	DriverApp
10	> SAMPL00000008	Business	Implementing	1 - High	driver given greenest possible route	DriverApp
11	> SAMPL00000001	Business	Releasing	1 - High	Can order a pizza on the mobile app UI	PizzaApp
12	> SAMPL00000004	Business	Releasing	1 - High	process credit card payment	PizzaApp

Figure 80 Editing Features Directly in Program Kanban

You'll notice there's a lot of other features available to you from the result. For example, you can change record states, which can be useful in a Kanban. With some tweaking of the query you would also be able to create new Stories and Tasks, clone records. Consult Compass documentation for more information.

9.2.2 Backlog

In HCL Compass EssentialSAFe, the backlog is a record state. Any work item in the backlog is in the **Backlog** state. The backlog queries show records that are in that state. For a [Program Backlog](#), these are features that are ready to be pulled into a Program Increment. For a [Team Backlog](#), these are stories just waiting to be worked on in an iteration. As with the Kanban queries, we can use inline-editing to size, prioritize and assign backlog items.

APITeam Team Backlog										
Inline Mode Action: Modify										
	id	Type	Priority	Headline	Priority	Owner	ProgramIncrement	Iteration	Solution	StoryPoints
1	> SAMPL00000013	User	1 - High	pull order from pizza app system	1 - High		PIZZA PI 1	PIZZA PI 1 Iter 1	PizzaDash	
2	> SAMPL00000020	Enabl	3 - Lo	design windows for facts	3 - Low		PIZZA PI 2		PizzaDas	
3	> SAMPL00000021	User	3 - Low	as a user i would like the app to remember my favorite pizza	3 - Low		PIZZA PI 1	PIZZA PI 1 Iter 3	PizzaApp	1 2
4	> SAMPL00000026	User	3 - Low	as a user i need to allow app to have access to phone dialer	3 - Low		PIZZA PI 2		PizzaDash	3 5 8 13 20 40 100

A Product Manager would manage a Program Backlog and a Product Owner would manage the Team Backlog. These are activities that would occur in planning sessions. While many of the record types reviewed in this document do have access control on them (see section 1), anyone can modify work items.

Because these queries are created in the Personal workspace, someone with Public Folder privileges would need to move them to a public team workspace so that the team-based queries are in a public location for easy access. A Product Owner might like to create a folder for their team to access team-specific information. To obtain Public Folder permissions, please reach out to your HCL Compass Administrator.

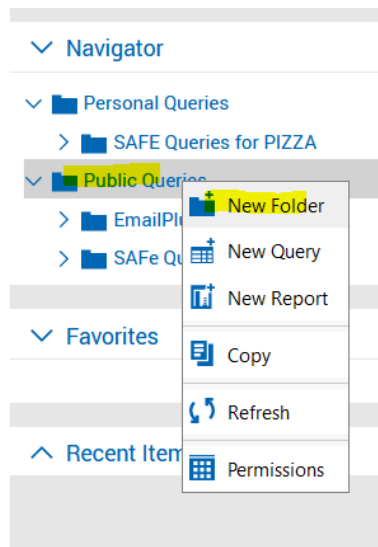


Figure 81 Creating a Folder in the Public Queries Workspace

10 EssentialSAFe – Schema or Package?

HCL Compass EssentialSAFe is available as either a schema or a package. Which method you use depends on your situation.

10.1 EssentialSAFe as a Schema

If you are starting with a new schema repository, this is the method you should use. Any new schema repository created with HCL Compass 2.0.1 will include the EssentialSAFe schema. To use the schema in that new schema repository, simply create a new user database. You can create a sample database with some example data in it when you create a new repository.

10.2 EssentialSAFe as a package

There are three situations in which you would use the EssentialSAFe package. In the first two instances you are trying to use EssentialSAFe in an existing Compass repository. In the third situation you may want to use different supporting packages. These cases are described below

10.2.1 Enabling a New Database

If you have a repository created with an older version of Compass, it does not already have the EssentialSAFe schema in it. If you would like to create a new user database for the EssentialSAFe schema, you would need to clone an existing schema, such as the Blank schema, and apply the EssentialSAFe and supplementary packages to it (see section 10.2.5). You can clone schemas and apply packages in the Compass Designer.

10.2.2 Enabling an Existing Schema

If you have an existing user database and you want to add EssentialSAFe capabilities to it, you can follow this approach. As a package, EssentialSAFe can be applied to an existing schema. We commend also applying the supplementary packages to it as described in section 10.2.5. Once the schema is enabled with these packages, upgrade the user database with the new version of the schema.

10.2.3 Using a Different Email, Notes or Attachment Solution

The EssentialSAFe schema comes pre-enabled with certain packages. These are described in section 8.4 and with more detail in section 10.2.5. You may choose to different solutions for these three capabilities. Perhaps you have developed your own in-house solution for notes, emails or attachments and want to use those instead. In this case you can apply the EssentialSAFe package, but then instead of applying the supplementary packages as described in section 10.2.5, you can apply your own solutions.

10.2.4 Dealing with Conflicts

An existing schema would already have record types defined and potentially other packages applied. During package application, the system will check for conflicts in record type names. It will not allow the package to be installed if any conflicts are found. If your existing schema is using any of the following record type names, they will need to be renamed prior to installation of the EssentialSAFe package:

- Component
- Feature
- Goal
- Iteration
- NFR

- Objective
- Persona
- ProgramIncrement
- Ranking
- Release
- ReleaseTrain
- Solution
- Stakeholder
- Story
- Task
- Team
- Teamiteration
- TeamPI

Note: when renaming record types, consider that your hooks may also need to be changed if they also reference these record types.

10.2.5 Supplementary Packages

When EssentialSAFe is applied as a package, you'll notice that some tabs you see in the EssentialSAFe schema are missing. This is because some of those tabs are package-owned themselves. Those [packages may have been applied](#) to the schema already or the Compass Administrator may prefer a different package or customization in its place.

The packages [applied](#) and [enabled](#) in the EssentialSAFe schema as seen in this document are:

- Notes 5.1 applied to Feature, Story, and Task record types
- EmailPlus 2.3 applied to Feature, Story and Task record types
- Attachments 1.0 applied to Feature, Story and Task record types

11 Permissions

The permissions model for the HCL Compass EssentialSAFe schema can be summarized in the table below. Anyone can create a ReleaseTrain and create the related records for it. But only an RTE or PM can create, modify or delete structural records associated with the release train. Structural records describe any record other than work items. Work items, such as Feature, Story, and Task can be created, modified, or deleted by anyone regardless of what ReleaseTrain they belong to.

Team, TeamPI, and TeamIteration records cannot be deleted, but they do have Delete actions. While only RTEs and PMs can create TeamPIs and TeamIterations, the PO for the associated Team can also modify these records.

All these permissions can be customized using optional global hooks described below.

RecordType	Create	Modify	Delete	MoveToRT ⁺
ReleaseTrain	Anyone	RTE, PM	RTE, PM	RTE, PM*
ProgramIncrement	RTE, PM	RTE, PM	RTE, PM	RTE, PM*
Iteration	RTE, PM	RTE, PM	RTE, PM	RTE, PM*
Team	RTE, PM	RTE, PM	None	RTE, PM*

TeamPI	RTE, PM	RTE, PM, PO	None	RTE, PM*
TeamIteration	RTE, PM	RTE, PM, PO	None	RTE, PM*
Solution	RTE, PM	RTE, PM	RTE, PM	RTE, PM*
Component	RTE, PM	RTE, PM	RTE, PM	RTE, PM*
Release	RTE, PM	RTE, PM	RTE, PM	RTE, PM*
Persona	RTE, PM	RTE, PM	RTE, PM	RTE, PM*
Objective	Anyone	Anyone	Anyone	Anyone
Feature	Anyone	Anyone	Anyone	Anyone
Story	Anyone	Anyone	Anyone	Anyone
Task	Anyone	Anyone	Anyone	Anyone

+ MoveToRT defines an operation where you are moving a record into another ReleaseTrain. How this is done depends on the record type. For example, on a ProgramIncrement you set a different ReleaseTrain. For a Component, this would be choosing a different Solution.

* Must be RTE or PM for both the source and destination Release Train.

12 Customization

The schema workflow can be customized to some extent. This can be done by defining some optional Perl subroutines in a custom global script. Each record type provided by HCL Compass EssentialSAFE has a base action with placeholders defined for all hooks. The hooks on these base actions will run any time an action is run on a record. The placeholders will look to see if certain Perl subroutines are defined and if they are it will call those subroutines.

12.1 Permission

If you would like to customize who can run which actions, you can define this subroutine:

```
$Permission = SAFECustomActionPermission ($entity, $Permission,
$actionname, $actiontype)
```

The arguments passed are the same as what a regular ACCESS_CONTROL hook would receive, \$entity, \$actionname, and \$actiontype. The HCL Compass Essential SAFE already has some built-in access control restrictions, for example who can modify a TeamPI record. The \$Permission variable passed in to SAFECustomActionPermission is the value calculating by built-in restrictions. You can choose to override this and grant or deny access. The custom permission subroutine should return the new permission value.

For example, the following subroutine would override all built-in access control and always allow the action to be run.

```
sub SAFECustomActionPermission($$$$) {
    my ($entity, $perm, $actionname, $actiontype) = @_;
    return 1;
}
```

12.2 Initialization

If you would like to add some initialization logic when actions are run, you can define this subroutine:

```
SAFeCustomActionInitialization($entity, $actionname, $actiontype);
```

The arguments passed are the same as what a regular INITIALIZATION hook would receive.

For example, the following subroutine calls a log statement every time an action is run on any record type (which may be useful for auditing or diagnostics).

```
sub SAFeCustomActionInitialization ($$$) {  
    my ($entity, $actionname, $actiontype) = @_;  
    my $s = $entity->GetSession();  
    my $msg = "Running $actionname on ".$entity->GetDisplayName();  
    $s->OutputDebugString($msg);  
}
```

12.3 Validation

If you would like to add some validation logic, you can define this subroutine:

```
$Validation = SAFeCustomActionValidation(  
    $entity, $Validation,  
    $actionname, $actiontype);
```

The arguments passed are the same as what a regular VALIDATION hook would receive, with the addition of the \$Validation parameter. The HCL Compass Essential SAFe already has some built-in validation logic, for example what ReleaseTrain you can set on ProgramIncrement record (you must be the RTE or PM for that ReleaseTrain). The \$Validation variable passed in to SAFeCustomActionValidation is the value calculating by built-in validation. You can choose to override this and perform different validation. The custom validation subroutine should return the new validation value. It should either be a string explaining why the validation failed, or an empty string to indicate successful validation. There are certain validation failures you cannot override. For example, if a value is not value because it isn't in a choice list, you cannot override it. The only validation failures this subroutine might see, and override, are ones described in the last column of the permission table above.

For example, the following subroutine essentially allows anyone to perform the MoveToRT described in the table above, e.g. move a PI to a different ReleaseTrain, even if they are not an RTE or PM for that ReleaseTrain.

```
sub SAFeCustomActionValidation ($$$$) {  
    my ($entity, $validation, $actionname, $actiontype) = @_;  
    return "";  
}
```

Commit

If you would like to add some Commit logic, you can define this subroutine:

```
SAFeCustomActionCommit($entity, $actionname, $actiontype);
```


The arguments passed are the same as what a regular COMMIT hook would receive. In this subroutine you could add code to modify related records and they would be in the same database transaction as the current record being modified. You could also choose to cancel the commit using a 'die' statement.

For example, the following subroutine lets you add custom commit logic for all actions.

```
sub SAFECustomActionCommit ($$$) {  
    my ($entity, $actionname, $actiontype) = @_;  
    # custom commit logic goes here...  
}
```

Notification

If you would like to add some Notification logic, you can define this subroutine:

```
SAFECustomActionNotification($entity, $actionname, $actiontype);
```

The arguments passed are the same as what a regular NOTIFICATION hook would receive. Notification hooks run after the database has been successfully updated. If you want to add integration to third party tool you could add notification hooks for that purpose.

For example, the following subroutine lets you add logic to update a third party integration for all actions.

```
sub SAFECustomActionNotification ($$$) {  
    my ($entity, $actionname, $actiontype) = @_;  
    UpdateMyThirdPartyTool($entity, $actionname, $actiontype);  
}
```

RecordScriptAlias Actions

The above mention optional Perl subroutines allow you to customize to the workflow of the HCL Compass EssentialSAFE schema to your liking. The exception to this, the RecordScriptAlias actions. These are special actions that do not trigger any base action hooks. The Utility menu actions are examples of RecordScriptAlias actions. The CreatePI action on the ReleaseTrain is one example of such an action that cannot be customized.

13 Best Practices

What follows are few recommendations for the best user experience when using HCL Compass EssentialSAFE.

- [HCL Compass EmailPlus package is enabled and configured on the user database](#)
- [HCL Compass Full Text Search is enabled for a richer user experience](#)