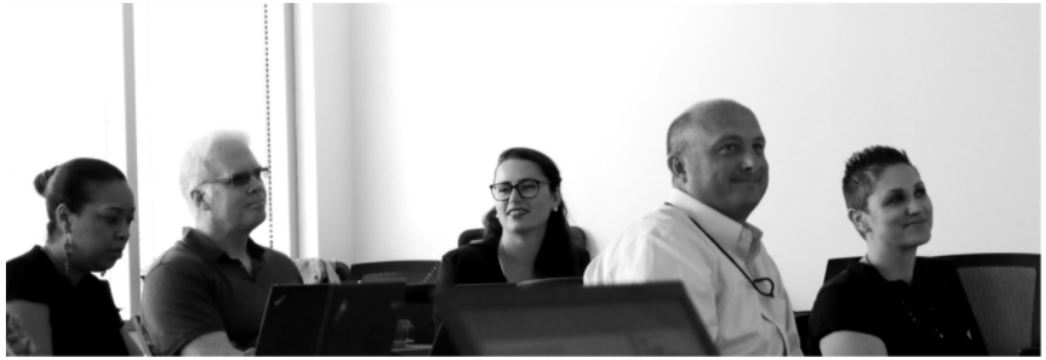
Notes

Introduction to Guidewire Configuration

Menu

Education

Introduction to Guidewire Configuration



NEXT ›

GUIDEWIRE

Notes

Fundamental areas of configuration

The three-tiers are the three fundamental areas of configuration.

There is also a fourth area of configuration for the development of integration points to external systems. Each application is typically integrated with a number of external systems. The connections to these systems are configured through the application APIs.

Typical examples:

- Policy Administration System
- Billing System
- Claim System
- Authentication (LDAP / Active directory)
- Document storage
- Document production
- Address Book (Contact management system)
- Printing System, etc....

Integration topics are discussed in Integration courses.

Fundamental areas of configuration

Database

Application Server

User Interface

Education

Notes

Data model entities

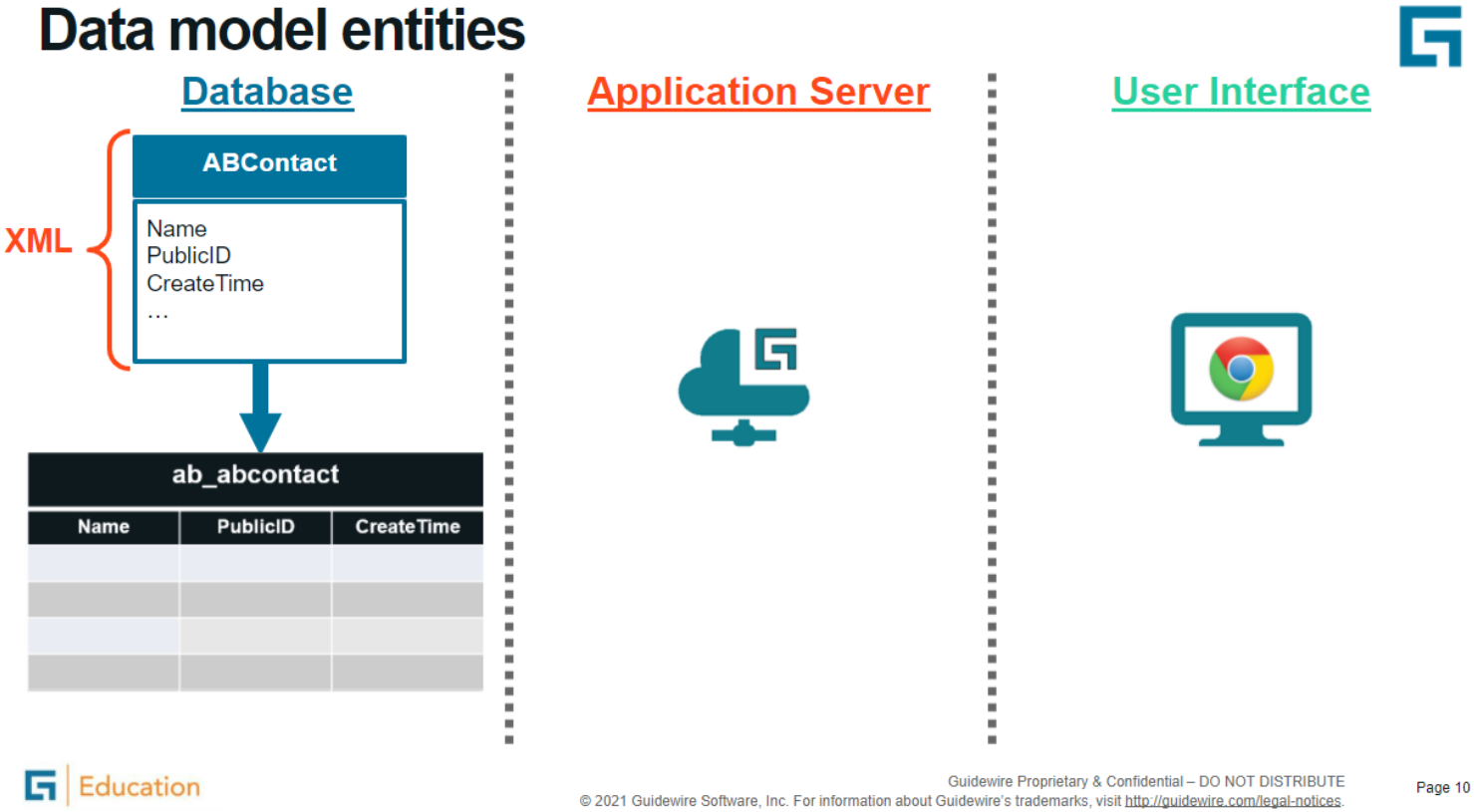
A data model entity is a high-level business object used by the Guidewire application, such as ABContact, Policy, Account, Invoice, Claim. It defines the information about the object that must be stored in the database, such as Name, PublicID, and CreateTime. It is defined in a set of one or more XML files.

In most cases, each data model entity corresponds to a table in the database. The data model entity definition defines the table structure. The fields of the entity will become columns in the database table. Each instance of the data model entity is stored as one row in the database table.

In development environments, when the application server gets restarted or the application gets re-deployed, the database will be automatically upgraded based on the data model.

Menu

Data model entities



Education

PREV NEXT

GUIDEWIRE

Notes

Gosu and generated Java classes

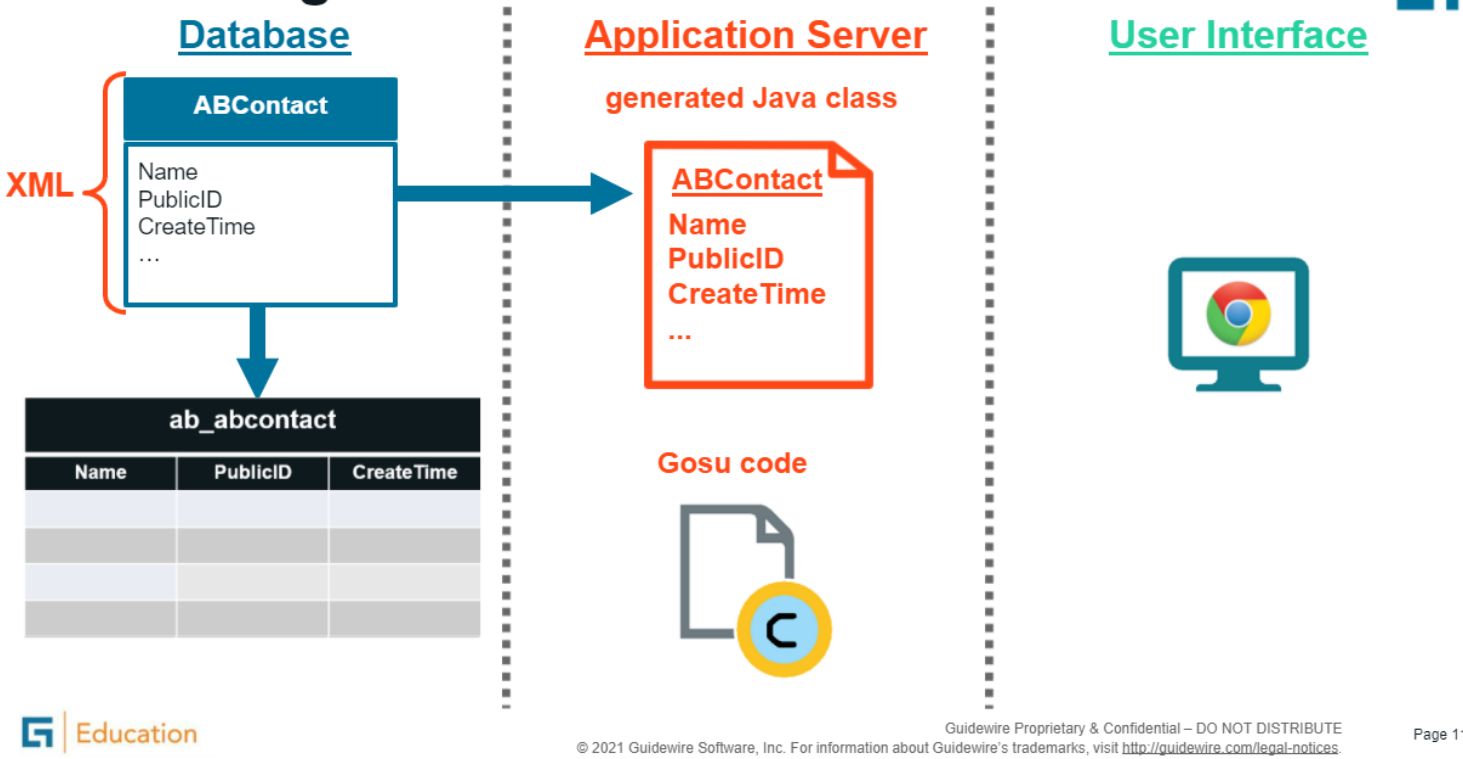
For every data model entity, there is a generated Java class with the same name. For every field in the data model entity, there is a field in the corresponding generated Java class. For example, the ABContact data model entity has a "Name" field, and the generated ABContact Java class also has a "Name" field. In InsuranceSuite, Guidewire Studio makes the process of code generation explicit and the generated code itself accessible. The generated Java class can be viewed and debugged in Studio.

Whenever the application needs to work with an instance of a data model entity (which is stored as a row in the corresponding database table), the application creates an instance of the corresponding Java class. The information from the database is then read into that instance. For example, if a user searches for the ABContact whose name is "Express Auto", then the application finds the row in the database table for ABContact, creates an instance of the ABContact Java class, and reads the data from that row into that instance.

The Application Server tier also contains Gosu resources, such Gosu classes, Gosu Rules, enhancements.

Menu

Gosu and generated Java classes



Education

PREV NEXT

GUIDEWIRE

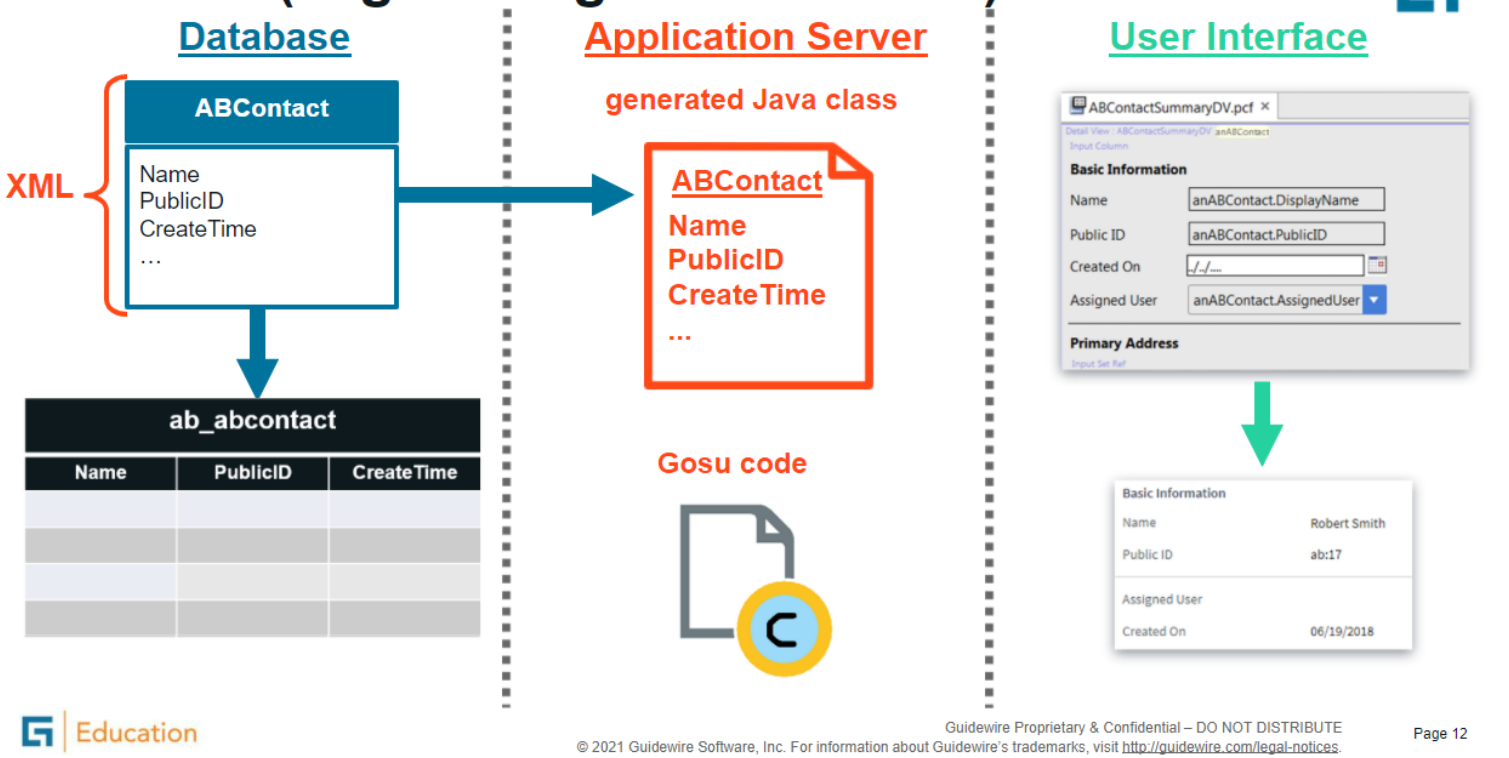
Notes

PCF files (Page Configuration Format)

A PCF file (Page Configuration Format) is an abstract definition of a form or location used by the user interface tier. For example, "ABContactSummaryDV" detail view is used to display information about an ABContact. It defines the information about the objects to be displayed in the user interface, such as the Name, Public ID, Created On and Assigned User fields. It is defined in a set of one or more XML files using the proprietary PCF XML schema definition.

There is a strong correspondence between data model entities and generated Java classes. Every data model entity has one generated Java class. There isn't necessarily a strong correspondence between generated Java classes and PCFs. The data for one Java class could be displayed in a single PCF, or it could be displayed across multiple PCFs. The separation of the user interface tier and the application server tier gives developers the freedom to display data in whatever way makes sense to end users without being constrained by how that data is maintained in the application server or how the data is stored in the database.

PCF files (Page Configuration Format)



Education

PREV NEXT

Notes

Components working together – displaying Express Auto (1 of 6)

In this example we can see how the system uses the different components to display data from the database.
The user clicks a button in the user interface to display the details of Express Auto.
TrainingApp performs the steps described on the following slides.

Components working together – displaying Express Auto (1 of 6)

Database

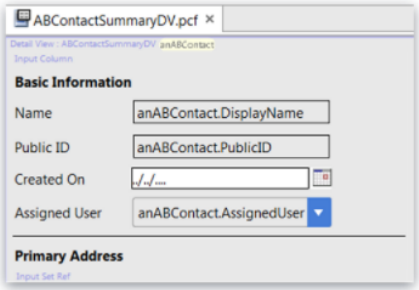
ab_abcontact

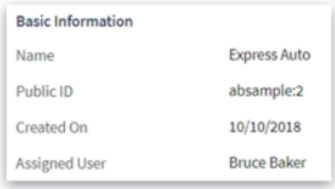
Name	PublicID	CreateTime
Express Auto	absample:2	10/10/2018
Burlingame Saab	ab:78	05/27/2018
Albertson's	ab:61	03/16/2018
Health South	ab:73	04/18/2018

Application Server

generated Java class

User Interface





Education

PREV NEXT

Notes

Components working together – displaying Express Auto (2 of 6)

1) First the system creates an instance from the appropriate generated Java class. The system knows that from the database table. This will be discussed in more detail in the Introduction to the Data Model lesson.

Menu

Components working together – displaying Express Auto (2 of 6)

Database

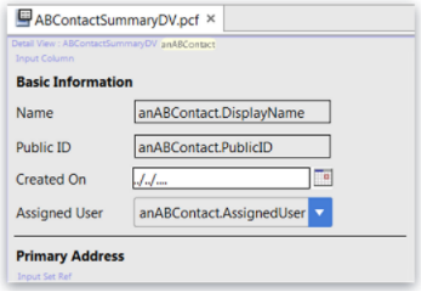
ab_abcontact

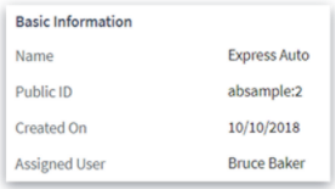
Name	PublicID	CreateTime
Express Auto	absample:2	10/10/2018
Burlingame Saab	ab:78	05/27/2018
Albertson's	ab:61	03/16/2018
Health South	ab:73	04/18/2018

Application Server

generated Java class

User Interface





Education

PREV NEXT

GUIDEWIRE

Notes

Components working together – displaying Express Auto (3 of 6)

2) The data from the row will be loaded into the object instance. Remember, for every column in the database table, there is a field in the corresponding generated Java class.

Menu

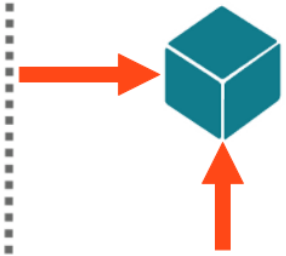
Components working together – displaying Express Auto (3 of 6)

Database

ab_abcontact

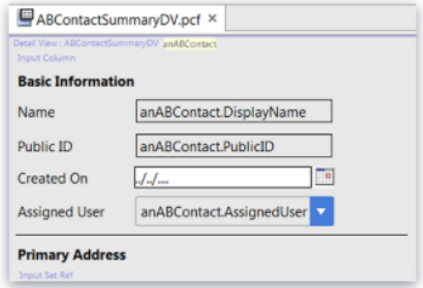
Name	PublicID	CreateTime
Express Auto	absample:2	10/10/2018
Burlingame Saab	ab:78	05/27/2018
Albertson's	ab:61	03/16/2018
Health South	ab:73	04/18/2018

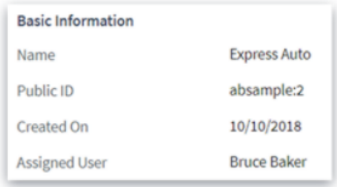
Application Server



generated Java class

User Interface





Education

PREV NEXT

GUIDEWIRE

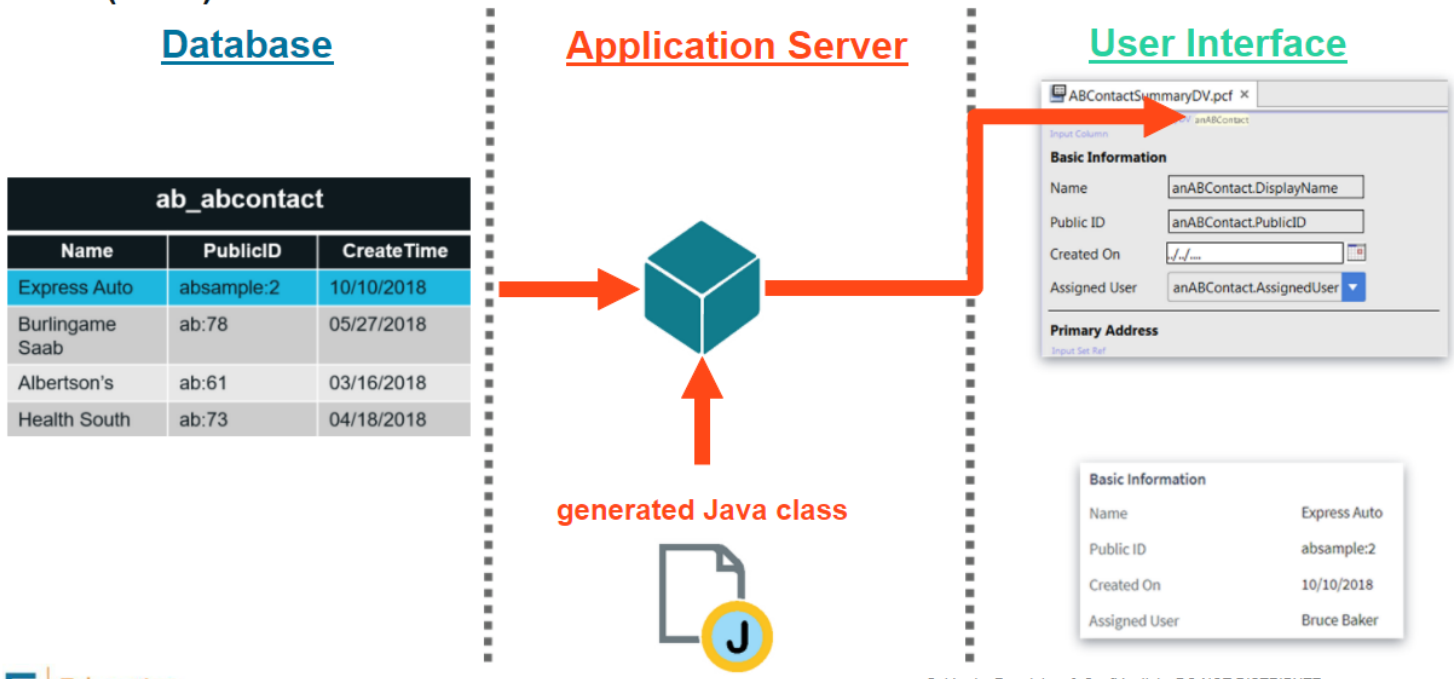
Notes

Components working together – displaying Express Auto (4 of 6)

3) Once the object is initialized it will be passed in as an input parameter to the ABContactSummaryDV PCF file. The object will be stored in the anABContact variable.

Menu

Components working together – displaying Express Auto (4 of 6)



Education

PREV NEXT

GUIDEWIRE

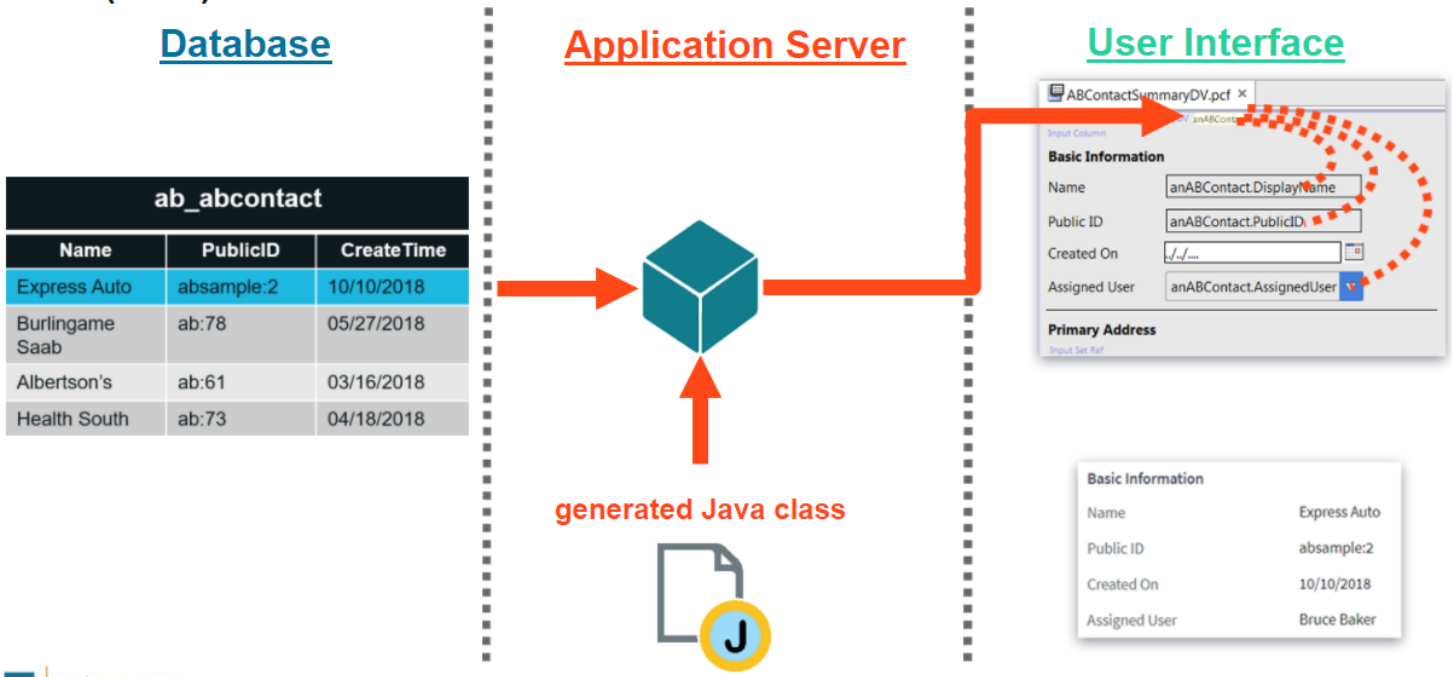
Notes

Components working together – displaying Express Auto (5 of 6)

4) The different PCF elements can use the dot notation to access the data in the object. You may be already familiar with the dot notation from other object oriented languages such as Java or C#.

Menu

Components working together – displaying Express Auto (5 of 6)



Education

Page 17

PREV NEXT

GUIDEWIRE

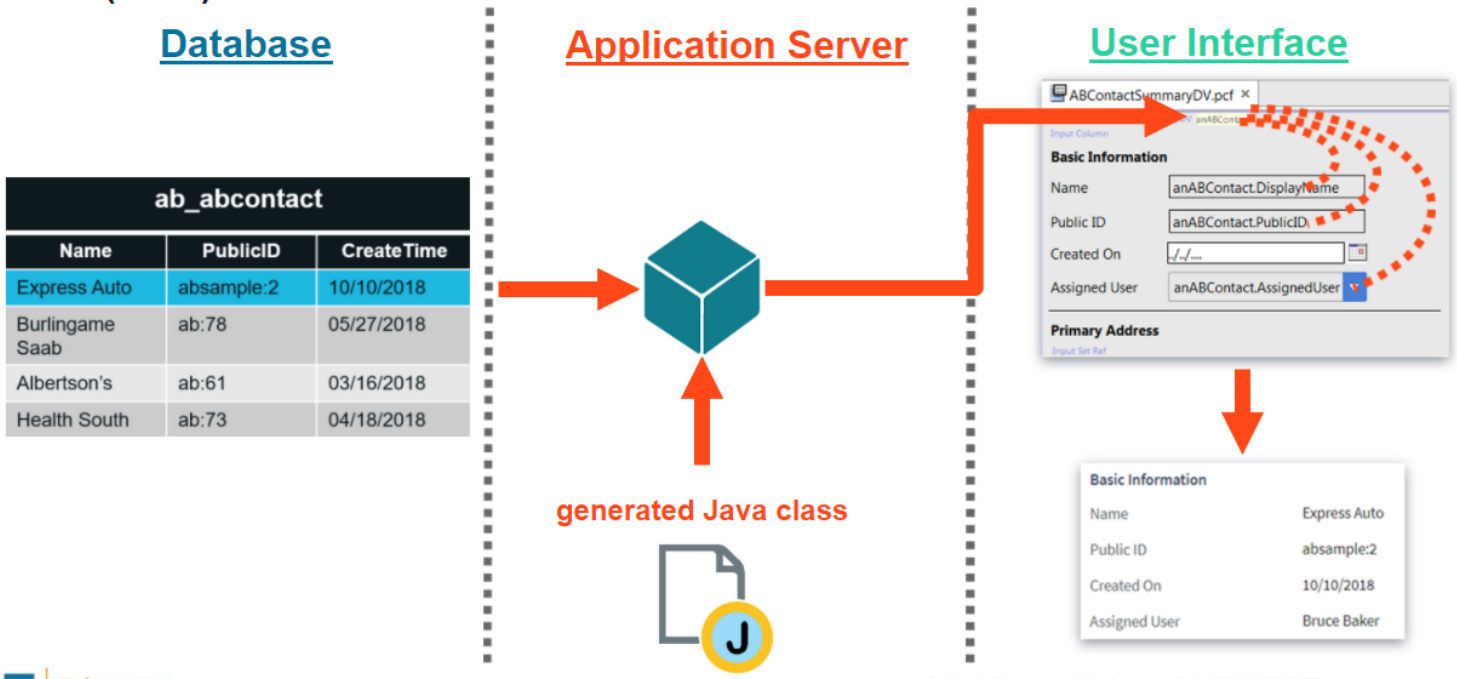
Notes

Components working together – displaying Express Auto (6 of 6)

5) TrainingApp generates the HTML page that will be rendered in the browser.

Menu

Components working together – displaying Express Auto (6 of 6)



Education

PREV NEXT

GUIDEWIRE

Notes

The Guidewire platform

The Guidewire platform is layer of configuration technology that includes the functionality needed to define a Guidewire applications. For example, it includes:

- The technology to define a data model
- The technology to define a user interface
- The technology to define application logic
- The technology to define integration points

Every application uses this common technology to define its own data model, user interface, business logic, and integration points. Each application is distinct, but every application shares common features and configuration techniques with all the other applications. For example, users can be created, modified or deleted the same way by the administrator in all the Guidewire applications. That is why data model entities, PCF files and application logic to manage Users are defined in the platform.

The Guidewire platform

- The Guidewire platform is a layer of configuration technology that includes the functionality needed to define a Guidewire application
- It contains common resources and configuration techniques that are shared by all Guidewire applications. For example:
  - Data model entities, PCF files, and application logic to manage Users
  - Data model entities and PCF files to create and assign Activities to other users
- This course focuses on the Guidewire platform

Education

Notes

Application-specific functionality

Each application also has application-specific functionality.

The **PolicyCenter-specific** functionality includes:

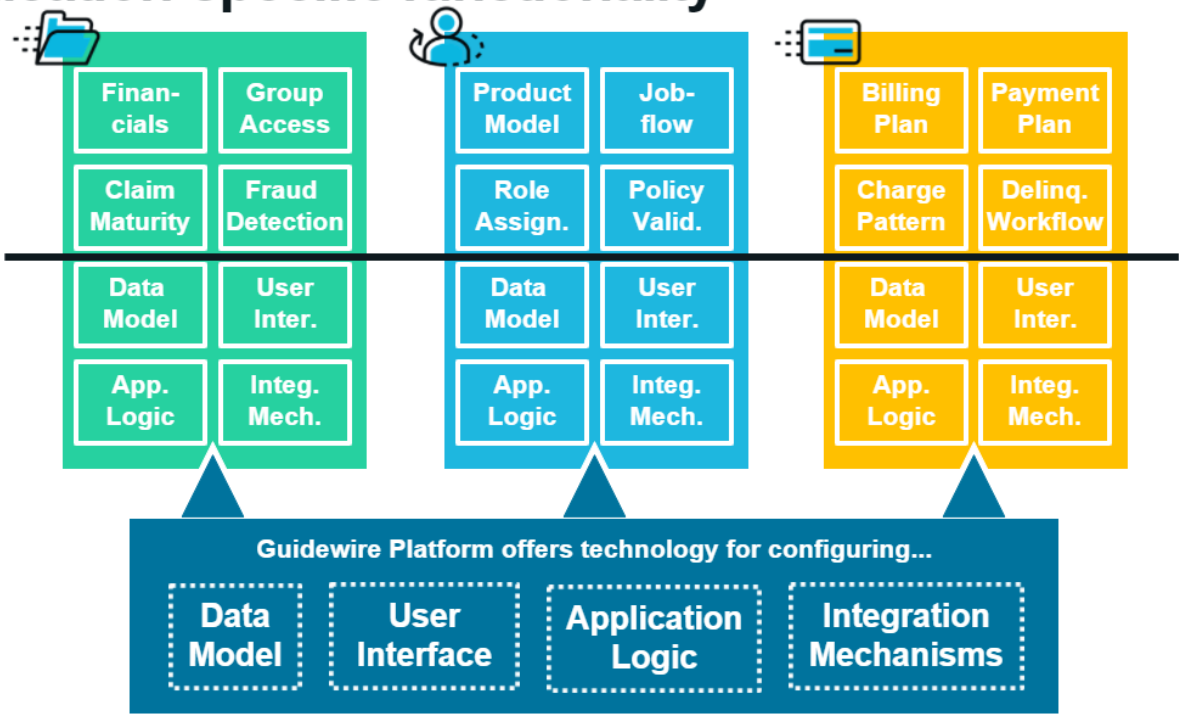
- The product model, which defines products, the policy lines assigned to each product, the coverages and coverables for the product, and the coverage terms for those coverages.
- Job flow, which defines how policy transactions (such as submissions, renewals, changes, and cancellations) are executed.
- Role assignment, which defines how users are assigned responsibility for a given account, policy, or policy transaction.
- Policy validation, which validates that a given policy is valid, quotable, bindable, or issuable.

The **BillingCenter-specific** functionality includes:

- The management of invoices over the billing cycle from planned to billed to due.
- The production of producer commission statements for business collected directly from the client and the production

Menu

Application-specific functionality



Education

PREV NEXT

GUIDEWIRE

Notes

Review

Menu

Review Questions

Education

PREV NEXT

Notes

How do you start Guidewire Studio?

Menu

Answer

How do you start Guidewire Studio?

Open the command line in the application folder and type “gwb studio”

Education

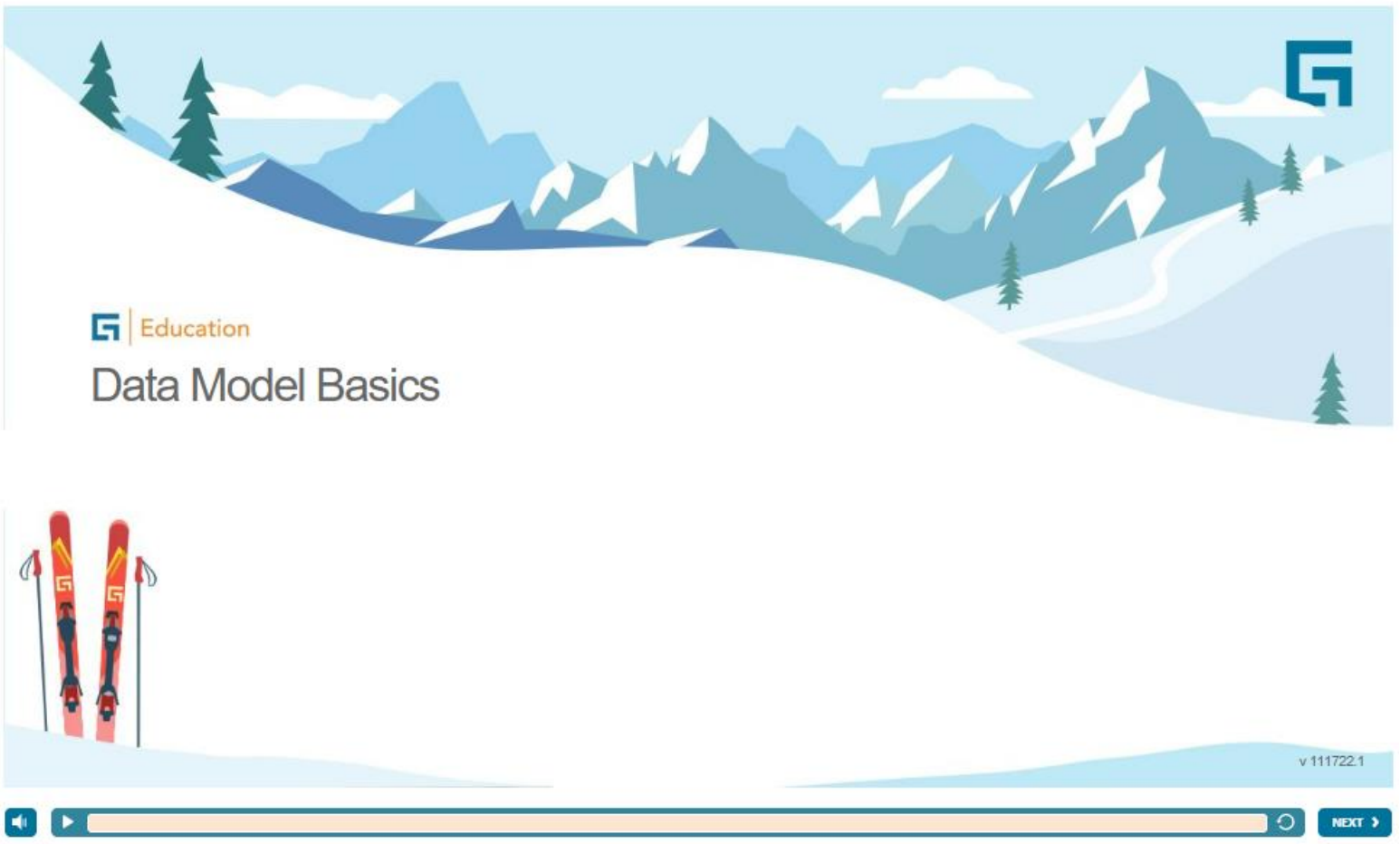
PREV NEXT

GUIDEWIRE

Notes

Data Model Basics

Welcome to the **Data Model Basics** module.



What is the data model?

A set of XML-formatted metadata definitions of entities and typelists

The core data structure for all Guidewire applications

Loaded on start-up of all Guidewire applications

Notes

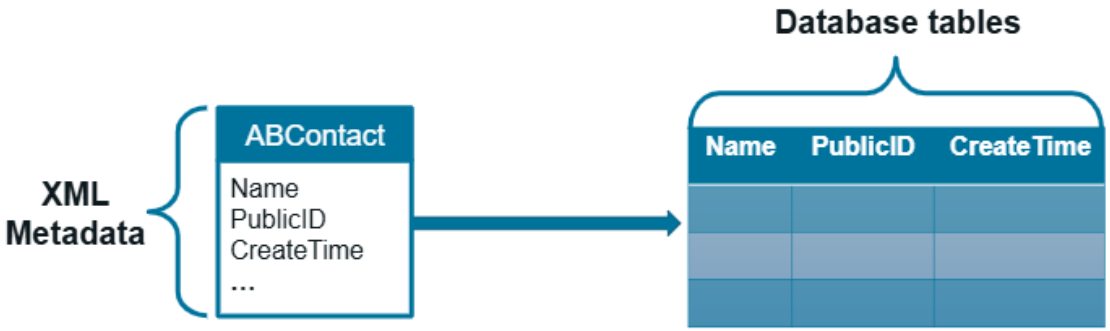
The Guidewire data model and the database

As part of the application start-up, the loaded metadata instantiates the data model as a collection of tables in the application database. Entities defined in the data model map to tables in the database and fields map to database columns, except for arrays and virtual properties.

Menu

The Guidewire data model and the database

The metadata instantiates a collection of tables in the application database



Education

PREV NEXT

Notes

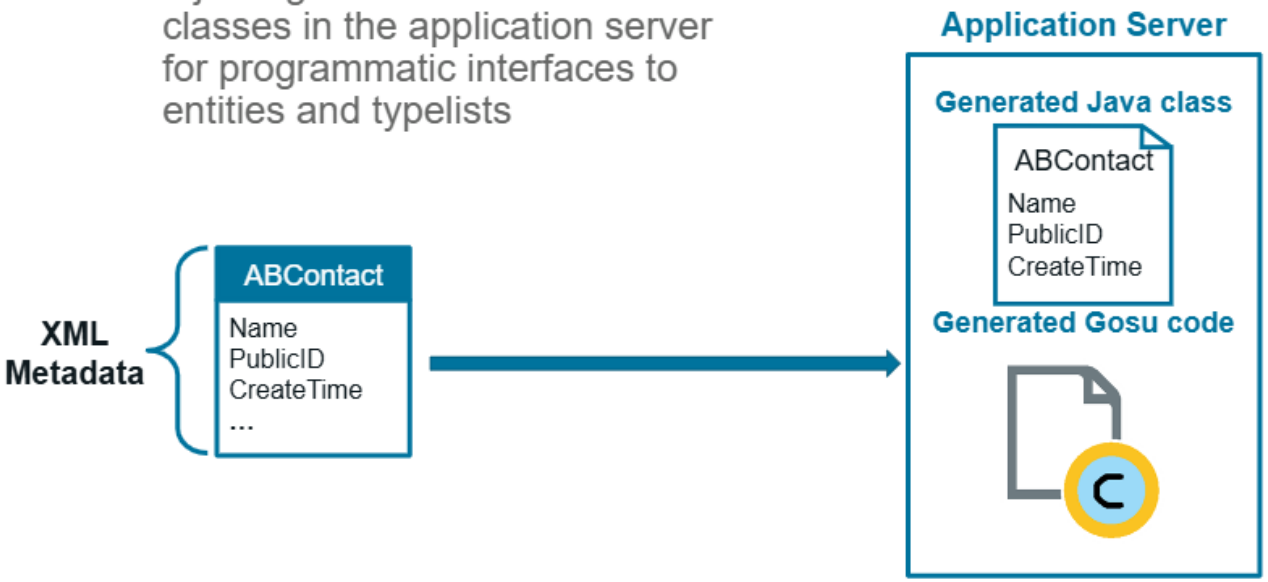
The Guidewire data model and classes

Guidewire applications employ a metadata approach to data objects. A Guidewire application uses the metadata about application domain objects to provide generated Gosu and Java interfaces to these objects. Those Java and Gosu classes provide programmatic access to the data objects defined in the data model.

Menu

The Guidewire data model and classes

Injects generated Java and Gosu classes in the application server for programmatic interfaces to entities and typelists



Education

PREV NEXT

GUIDEWIRE

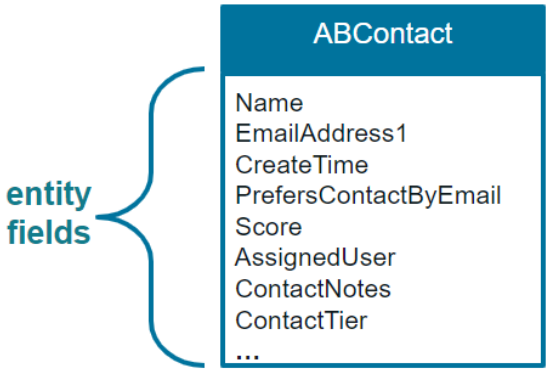
Notes

Data fields

Data fields are defined in XML for an entity file with a <column> element. Therefore, they are sometimes referred to a "column fields". There are other types of fields beyond primitive value fields that are stored in database table columns, however. So one should not assume that the only fields that map to database columns are the "column fields". This course uses the term "data fields" to avoid this possible point of confusion.

Data fields

- A data field stores a single value that does not reference any other object or table
- Typically defined with the `<column>` element
- Examples of single values:
  - Name is a varchar
  - CreateTime is a datetime
  - PrefersContactByEmail is a bit
  - Score is an integer



Education

Notes

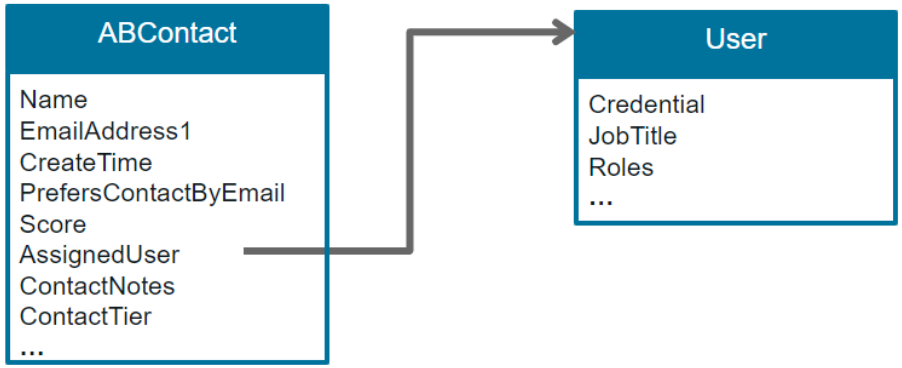
Foreign key fields

Note: At its simplest, the Guidewire data model is a set of XML-formatted metadata definitions of entities and typelists. An entity defines a set of fields for information. You can add a Foreign Key field to an entity.

Menu

Foreign key fields

- A foreign key field stores a reference to a related object in the Data Model
- It defines a unidirectional relationship
- AssignedUser in ABContact is the foreign key field for a User



Education

How to define an Array relationship between two Entities A and B ?

Notes

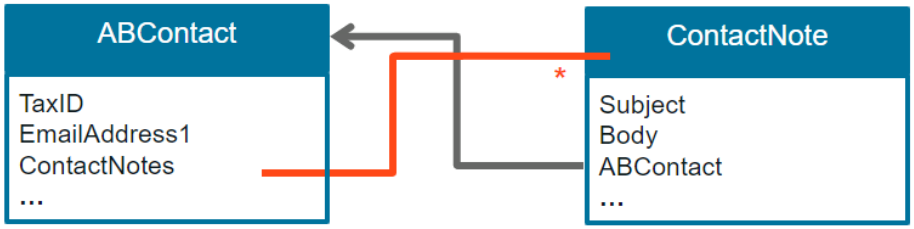
Array fields

Note: At its simplest, the Guidewire data model is a set of XML-formatted metadata definitions of entities and typelists. An entity defines a set of fields for information. You can add an Array field to an entity.

Menu

Array fields

- An array defines a set of additional entities of the same type to associate with the main entity
- For example, ABContact entity includes an array of ContactNote entities. This means ABContact can have zero, one or more ContactNotes associated to it.



Education

What is Typelist and what entity property we use to save in the Entity ?

Notes

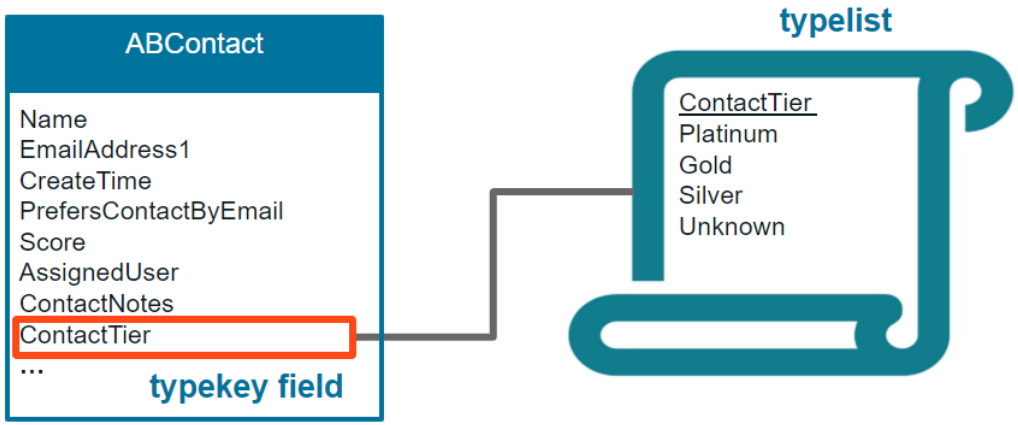
Typelists and typekey fields

Note: At its simplest, the Guidewire data model is a set of XML-formatted metadata definitions of entities and typelists. An entity defines a set of fields for information. You can add a Typekey field to an entity.

Menu

Typelists and typekey fields

- A typelist is a predefined set of possible values that limits the acceptable values for fields within the application. In most cases a typelist is rendered as a drop-down list in the user interface
- A typekey field is entity field that is associated with a specific typelist. It can point to exactly one value in the typelist



Education

PREV NEXT

GUIDEWIRE

Notes

Supertype (subtyped) entities

Subtype entities are children entities of a top level supertype (parent) entity. All fields in the parent entity are inherited by the child entity.

ABContact is a top level supertype entity. The ABContact entity is an abstract entity (abstract=true). Abstract entities cannot be instantiated, but they can be subtyped. In other words, this means that you cannot create instances of an ABContact, but you can create entity instances of its subtypes such as instances of ABPerson, ABCompany, ABPersonVendor, ABPolicyPerson, ABAjudicator, ABAttorney, and ABDoctor.

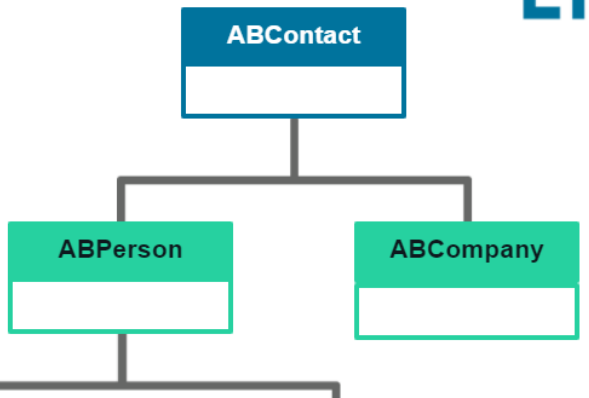
Not all top level supertype entities are abstract. It is also possible to have subtype entities that are abstract

Another way to describe a supertype entity is to use the term "subtyped" entity. In the slide example, there are several subtyped entities including ABContact, ABPerson, and ABPersonVendor. The slide diagram does not reflect all the supertype and subtype entities in TrainingApp.

Supertype entities in **ClaimCenter** include Contact, Incident, and Transaction. Contact and Transaction are abstract entities

Supertype (subtyped) entities

- If multiple entities have a common set of fields then they could be organized into a hierarchy
- The entities in the same hierarchy will use the concept of inheritance
- This means that fields and methods of supertype entities are inherited by subtype entities



Education

PREV NEXT

How does Entity gets saved in the underlying database ?

Notes

Entities in the database

There are two exceptions to the generalization above: virtual and subtype entities.

Virtual. Non-persistent entities are entities constructed entirely with code. They exist during run-time, and when the Guidewire application shuts down, the data is not saved to the database. An example of a non-persistent entity is ABContactSearchCriteria in TrainingApp. Guidewire recommends that you do not create non-persistent entities as a general rule

Subtype entities are entities that are children entities of a top level supertype (parent) entity. All fields in the parent entity are inherited by the child entity. A top level supertype entity and all of its child subtypes are stored in a single database table. In this manner, all the entities share the same database table.

Menu

Entities in the database

Table - dbo.ab_abcontact*

	ID	Name	CreateTime	PrefersContact...	Score	AssignedUser
	64	United Natural Foods Inc	9/4/2009 1:51:32 PM	False	*NULL*	2
	65	3M	9/4/2009 1:51:32 PM	False	*NULL*	3
	74	Express Auto	9/4/2009 1:51:32 PM	False	81	3
	75	European Autoworks	9/4/2009 1:51:32 PM	True	*NULL*	4
	76	M B Garage	9/4/2009 1:51:32 PM	False	*NULL*	7
	77	Burlingame Saab	9/4/2009 1:51:32 PM	False	*NULL*	2

ABContact

Name
EmailAddress1
CreateTime
PrefersContactByEmail
Score
AssignedUser
ContactNotes
ContactTier
...

- Often, an entity stores data in a database table named for the entity:
  - ABContact entity in ab_abcontact table
- Exceptions are…
  - Entities that share the columns in an entity table such as a subtype entity
  - Virtual, non-persistent entities do not save data to the database. Guidewire recommends that you do not create non-persistent entities as a general rule

Education

PREV NEXT

Notes

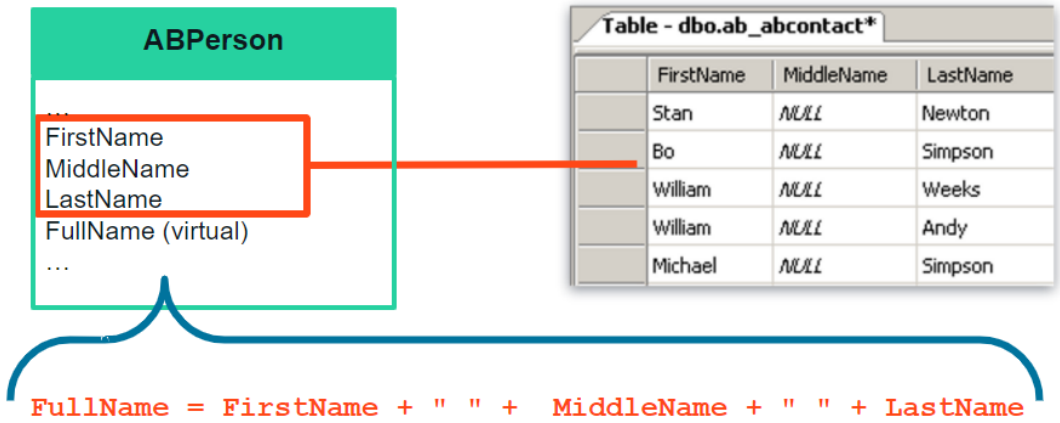
Data fields in the database

In the example above, the FullName field is a virtual field or property. FullName concatenates the values of the FirstName, MiddleName, and LastName physical fields.
Virtual fields are defined as Gosu code in Enhancements. Entity Enhancements are discussed in another lesson.

Menu

Data fields in the database

- Data fields can be physical or virtual
  - Data model defines physical fields as columns in a database table
  - Gosu code defines virtual fields and there is no physical database column



Education

PREV NEXT

GUIDEWIRE

Foreign key connected PK of one table to FK for another table

Notes

Foreign key fields in the database

In the slide example, the assigned user (4) for European Autoworks is Alice Applegate. The assigned user (3) for 3M and Express Auto is System User. The assigned user (2) for United Natural Foods Inc is Super User.

Menu

Foreign key fields in the database

- Foreign key fields are stored as foreign key columns

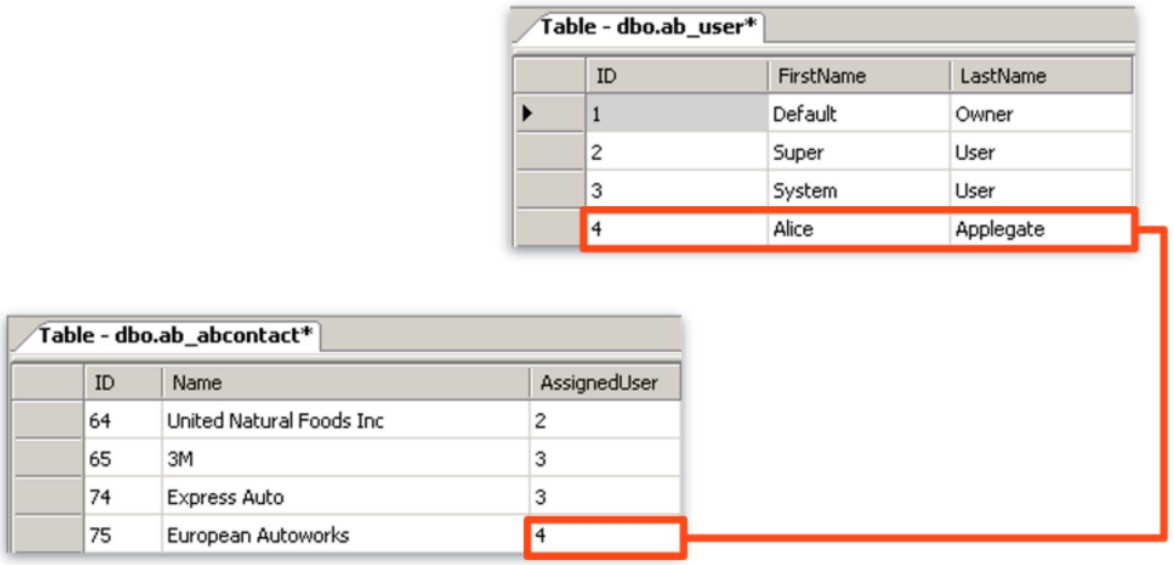


Table - dbo.ab_user*

	ID	FirstName	LastName
▶	1	Default	Owner
	2	Super	User
	3	System	User
	4	Alice	Applegate

Table - dbo.ab_abcontact*

	ID	Name	AssignedUser
	64	United Natural Foods Inc	2
	65	3M	3
	74	Express Auto	3
	75	European Autoworks	4

Education

In Arrays – We create array from one Entity and reverse Foreign key from other Entity

Notes

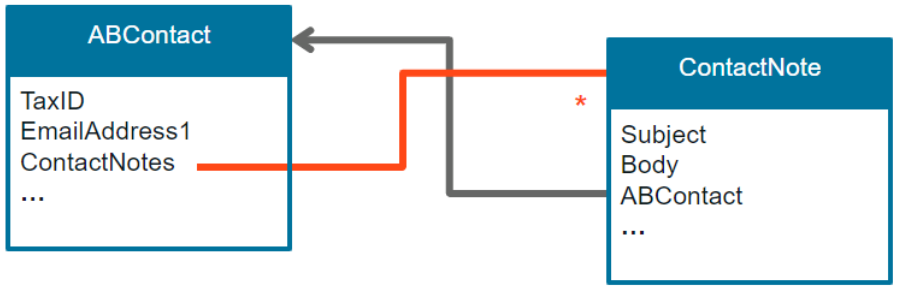
Array fields in the database

Array keys are managed during run-time by code that queries the database for objects of a given type with foreign keys that point to the array's parent.

For example, assume you have an ABContact object named ericAndy that stores the ABContact data for Eric Andy. Whenever you reference ericAndy.ContactNotes, Guidewire queries the database for all ContactNote objects whose ABContact field points to Eric Andy and returns those objects in an array.

Array fields in the database

- Arrays are not stored in the database, but populated at runtime by queries
- Every array on the main entity must have a reverse foreign key pair on the associated additional entity



Education

Notes

Typelists and typekey fields in the database

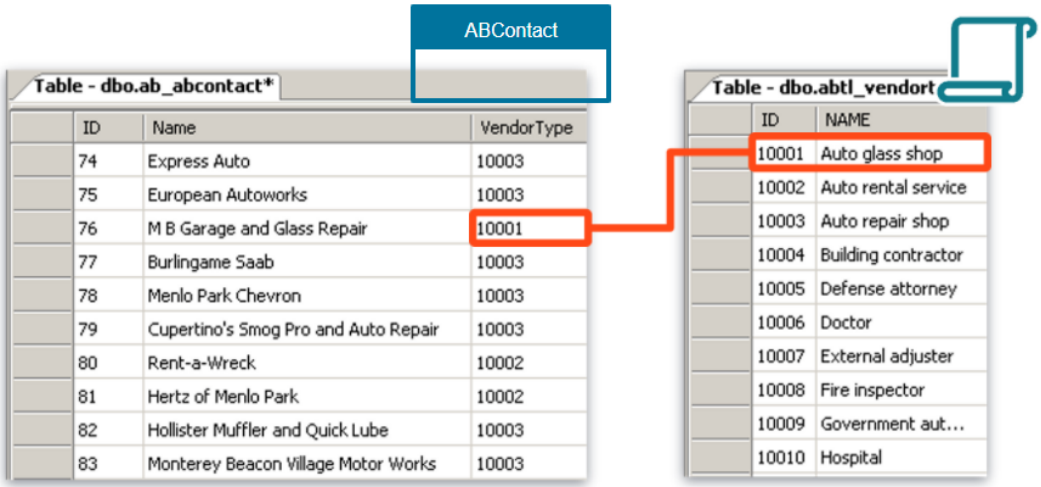
Typelist tables are named <applicationcode>tl_<typelistname>.

In this example, the VendorType of M B Garage and Glass Repair is Auto glass shop.

Menu

Typelists and typekey fields in the database

- Each typelist is stored in its own table
  - This is a predefined, fixed table that does not change during runtime
- Each typekey is a foreign key to a single row in a typelist table



Education

PREV NEXT

GUIDEWIRE

How does Subtype gets saved in the database ?

how to retrieve subtype from the database ?

Notes

Subtype entities in the database

Whenever an entity is subtyped, a virtual typelist is automatically created for it. This typelist contains one typecode for every subtype of the parent entity. Typelist tables are prefixed with "xxtl_", where "xx" is the two-letter application code.

The ab_abcontact table is the parent or supertype table that stores all instances of ABContact and its subtypes.

Menu

Subtype entities in the database

- Supertype parent table stores all instance of itself AND of subtype data
  - Contains parent fields and all subtype fields
  - Irrelevant fields are null for specific subtypes
  - Application automatically creates typelist table of all subtypes
  - Subtype column identifies subtype
- Example: FIRSTNAME and LASTNAME are always null for subtype = 3

SELECT ID, TYPECODE
FROM ABTL_ABCONTACT

ID	TYPECODE
1	ABAdjudicator
2	ABAttorney
3	ABAutoRepairShop

SELECT ID, TAXID, NAME, FIRSTNAME, LASTNAME, VENDORTYPE, SUBTYPE FROM AB_ABCONTACT order by SUBTYPE;

ID	TAXID	NAME	FIRSTNAME	LASTNAME	VENDORTYPE	SUBTYPE
71	3432-32-431134-23-2343	*null*	James	Smythe	*null*	1
72	*null*	*null*	Paul	Peterson	*null*	1
67	2096542-3113-2456902	*null*	Lily	Watson	*null*	2
68	9983200-5335-0023899	*null*	James	Andersen	*null*	2
75	1242577-7777-7752421	Express Auto	*null*	*null*	*null*	3
76	3219251-8888-1529123	European Autoworks	*null*	*null*	*null*	3
77	5040392-9999-2930405	M B Garage	*null*	*null*	*null*	3
78	5647382-7777-2837465	Burlingame Saab	*null*	*null*	*null*	3
79	5646372-4444-2736465	Menlo Park Chevron	*null*	*null*	*null*	3

Education

PREV NEXT

GUIDEWIRE

Define an Entity ? Any Example

Notes

Entity

At its simplest, the Guidewire data model is a set of XML-formatted metadata definitions of entities and typelists. An entity defines a set of fields for information.

Menu

Entity

object entity

- An entity is a definition of a business object used by the application, such as Policy, User, ABContact, Producer, etc.
- Entities are defined in entity definition files (ETI)
  - The ETI file describes the entity in XML format
- There are two main types of entities:
  - Base application entities
  - Custom entities

Education

PREV NEXT

GUIDEWIRE

What are .eti ?

Notes

Base application entities

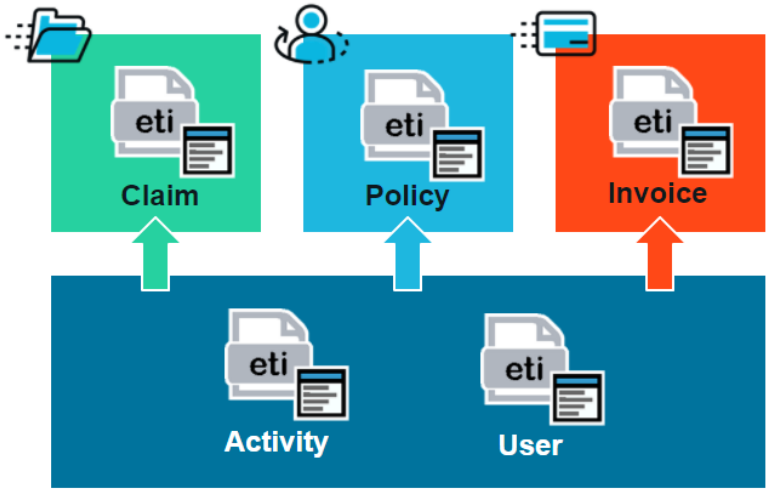
Platform entities are always found in the ... \modules\configuration\config\metadata\entity\ folder. All ETI files in the \metadata\entity\ are read-only. You cannot edit these files directly in Guidewire Studio and should not edit these files in any other application. Many platform entities have the platform="true" attribute defined in the <entity /> element.

Some application specific entities can also be found in the \metadata\entity\ folder. Many application entities are extendable in the sense that they can be both subtyped and/or an entity extension can be created. Entities with the final="true" attribute defined in the <entity /> element cannot be subtyped. Entities with the extendable="false" attribute defined in the <entity /> element cannot be extended in the sense that new elements cannot be added to the entity.

Menu

Base application entities

- A base application entity is an entity created by Guidewire and shipped as part of the out-of-the-box configuration
- Guidewire creates base application entities on the application or the platform level



- Application entities are specific to application
- Platform entities are common to all Guidewire applications

Education

PREV NEXT

GUIDEWIRE

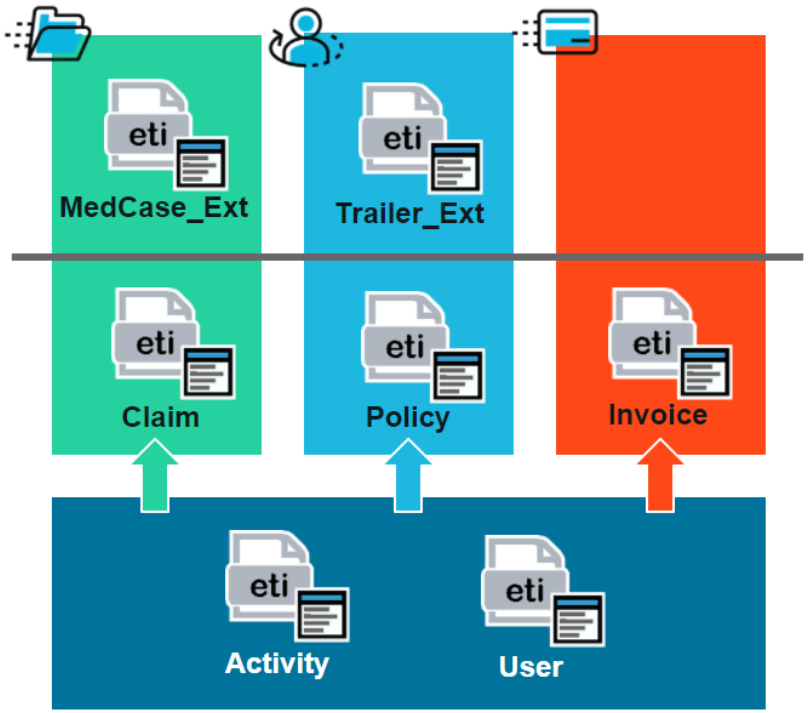
Notes

Custom entities

Custom entities are always found in the … \modules\configuration\config\extensions\entity\ folder.

Menu

Custom entities



- A custom entity is an entity created by the customer
- A custom entity can be created only on the application level and it will be local inside that application

Education

PREV NEXT

GUIDEWIRE

Notes

Entity extension

Note: To avoid future naming conflicts when Guidewire adds or changes base entities, Guidewire recommends that you add the suffix _Ext to your entity names and new field names on base entities. Adding the _Ext suffix to entity names results in Studio and the Data Dictionary listing these extensions next to any entities that they extend. For example, CreditHistory_Ext.

If you add a new entity, its field names do not need the _Ext suffix in their names, because you have the _Ext suffix in the name of the entity.

Entity extension

- Base application entities are read-only to prevent any conflicts during upgrade to the next version of the Guidewire application
- The concepts of an Entity extension allows customers to safely add new fields to base application entities or to override certain attributes of existing base application entity fields
- The entity extension is defined in an ETX file
- To keep the maintenance of entities simple a base application entity can have at most one ETX file
- For fields that are added to a base application entity, Guidewire recommends that the field name should end with _Ext

Notes

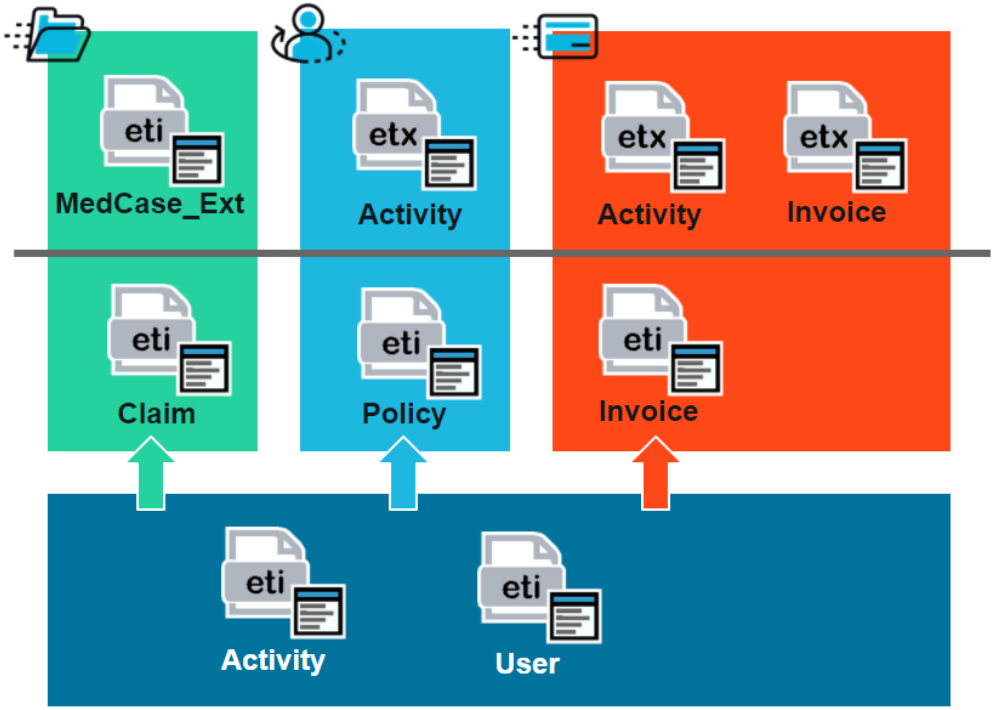
Extending base application entities

Customers can create or edit entity extensions. Entity extensions are always found in the … \modules\configuration\config\extensions\entity\ folder.

Guidewire provides certain entity extensions as part of the base application configuration. Many of the extension index definitions address performance issues. Other extensions provide the ability to configure the data model in ways that would not be possible if the extension was part of the base data model. Do not simply overwrite a Guidewire extension with your own extension without understanding the full implications of the change.

Menu

Extending base application entities



- The ETX file can be created on the application level and it is local inside that application

Education

PREV NEXT

GUIDEWIRE

What are .etx ?

Notes

Internal Entity Extension

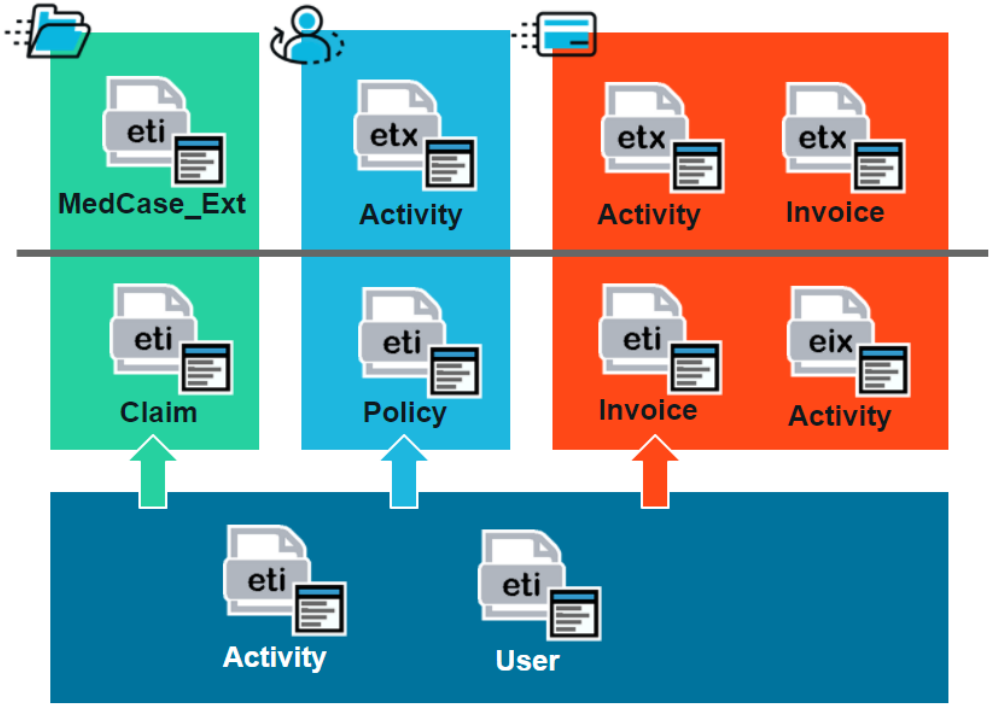
The EIX file contains extensions to the platform-level entities that are required for the base application and cannot be modified. EIX files are extensions to platform-layer entities created by Guidewire development to meet the needs of a given application's base data model.

EIX files are always found in the ...\modules\configuration\config\metadata\entity\ folder. All ETI and EIX files in the \metadata\entity\ are read-only. You cannot edit these files directly in Guidewire Studio and should not edit these files in any other application.

Because EIX files are neither created nor modified by configuration developers, this lesson does not discuss their structure. If you need to make multiple extensions to a single entity at different points in time, all extensions for that entity should be added to the same ETX file.

Menu

Internal Entity Extension



- An Internal Entity Extension is created by Guidewire to add application specific functionality to a platform level entity
- An EIX file is local in the application and read-only

Education

Page 9

PREV NEXT

Best Practices to add a new column in existing entity ?

Notes

Entity extensions and Java classes

For a base application data model entity, the entity is defined by both its original ETI file and the ETX extending file, if one exists. In certain cases, there is an internal entity extension, EIX, file. Regardless of which file the field is declared in, all fields from all files become the generated Java class.

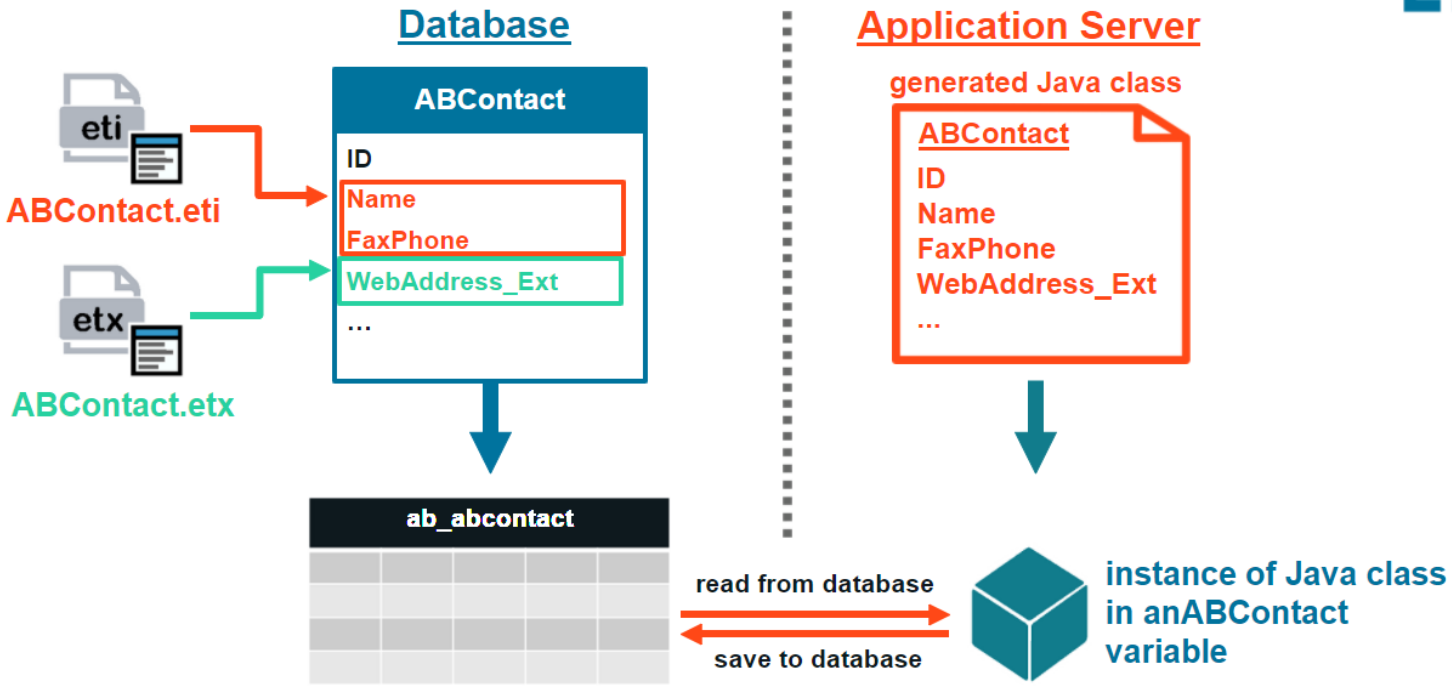
For fields that are added to a base application entity, Guidewire recommends that the field name should end with the suffix _Ext. In the slide example, the ABContact.etx entity extension file contains the WebAddress_Ext field.

Students coming from an Object-Oriented Programming (OOP) background should be aware that the term "extend" gets used in OOP differently than it does in Guidewire. In OOP, the term "extend" is used to refer to creating a new subclass that extends some parent superclass.

In Guidewire data model configuration, the term "extend" is sometimes used to refer to adding new fields to an existing base application entity. In this sense, no new subclass or subtype entity is getting created. One is simply adding additional fields to an existing base application entity.

Menu

Entity extensions and Java classes



Education

PREV NEXT

What is an Typelist, How do we define one ?

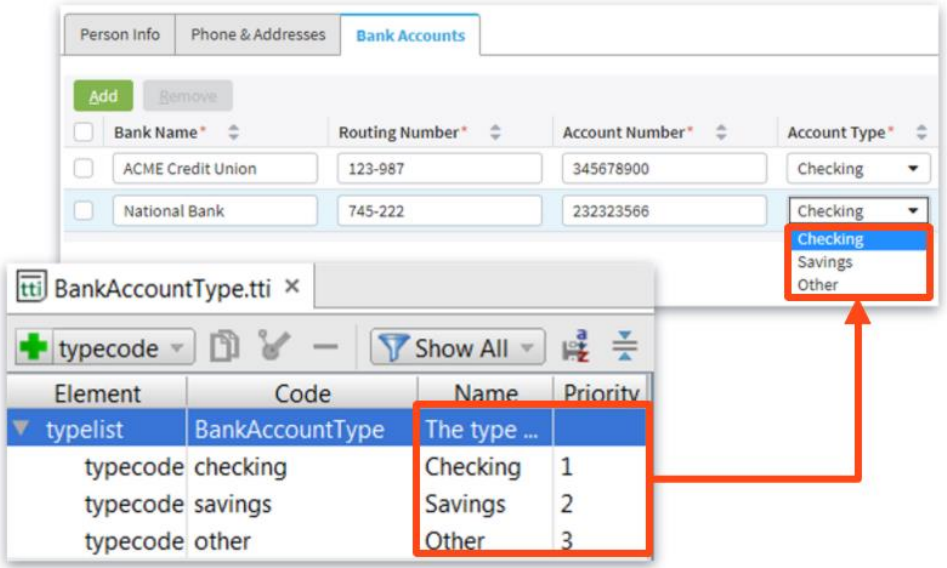
Notes

Typelist

You can programmatically get a list of typekeys in a typelist. You can choose whether to get all typekeys, including retired typekeys as well as non-retired typekeys, or just non-retired typekeys. You can retire a typekey in the data model configuration files. To get typekeys from a typelist, call the getTypeKeys method: TYPENAME.getTypeKeys(includeRetiredTypekeys)

Typelist

- A Typelist is a fixed, predefined list of values (Typecodes)
  - Values cannot be dynamically added or removed during runtime
- Defined value often used to constrain user input
  - When rendered in UI, typically appears as a dropdown list



Education

PREV NEXT

Notes

Typelist extension

Filetypes are defined as follows:

.tti - Typelist Type Info, a single Guidewire or custom typelist declaration. The name of the file corresponds to the name of the typelist being declared.
.tix - Typelist Internal extension, a single Guidewire typelist extension. The name of the file corresponds to the name of the Guidewire typelist being extended.
.ttx - Typelist Type extension - a single Guidewire or custom typelist extension. The name of the file corresponds to the name.

Typelist extension

- Base application Typelists are read-only to prevent any conflicts during upgrade to the next version of the Guidewire application
- The concepts of a Typelist extension allows customers to safely add new Typecodes to base application Typelists
- The Typelist extension is defined in a TTX file
- To keep the maintenance of Typelists simple a base application Typelist can have at most one TTX file
- For Typecodes that are added to a base application Typelist, Guidewire recommends that the code should end with _Ext

Education

Notes

What is the difference between an ETX file and an ETI file? (Answer)

Menu

Answer

What is the difference between an ETX file and an ETI file?

An ETX file extends a base application entity e.g. adding new columns. An ETI file defines a new custom entity from scratch.

Education

Notes

Does the base application have ETI files? Can developers create new ETI files? Where and how? (Answer)

Menu

Answer

Does the base application have ETI files? Can developers create new ETI files? Where and how?

Yes, ETI files are in the base application. Developers can create new ETI files in the …\extensions\entity\ folder using Guidewire Studio and the Entity Editor.

Education

PREV NEXT

GUIDEWIRE

Entities are XML formatted meta-date definitions

Notes

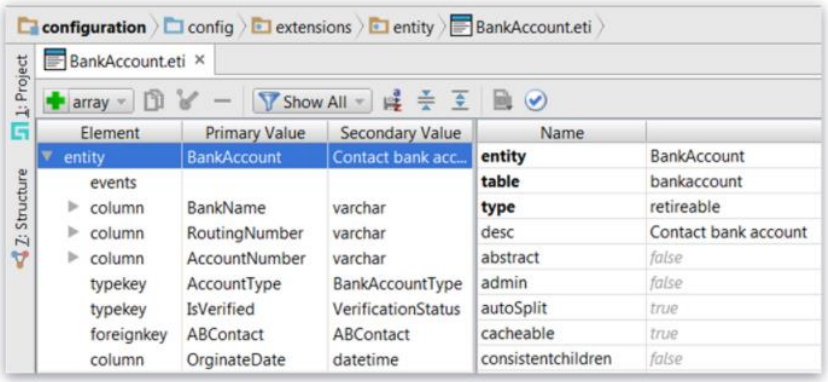
Review: Entities in the data model (1 of 2)

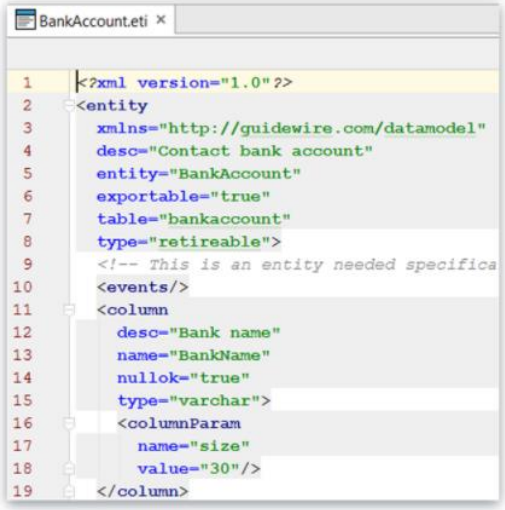
Note: All InsuranceSuite application objects exist as one of the base data objects or as a subtype of a base object.

Menu

Review: Entities in the data model (1 of 2)

- An entity is a business object used by the application, such as Policy, User, ABContact, Producer, etc.
- Entities are defined in ETI files





Education

Page 6

PREV NEXT

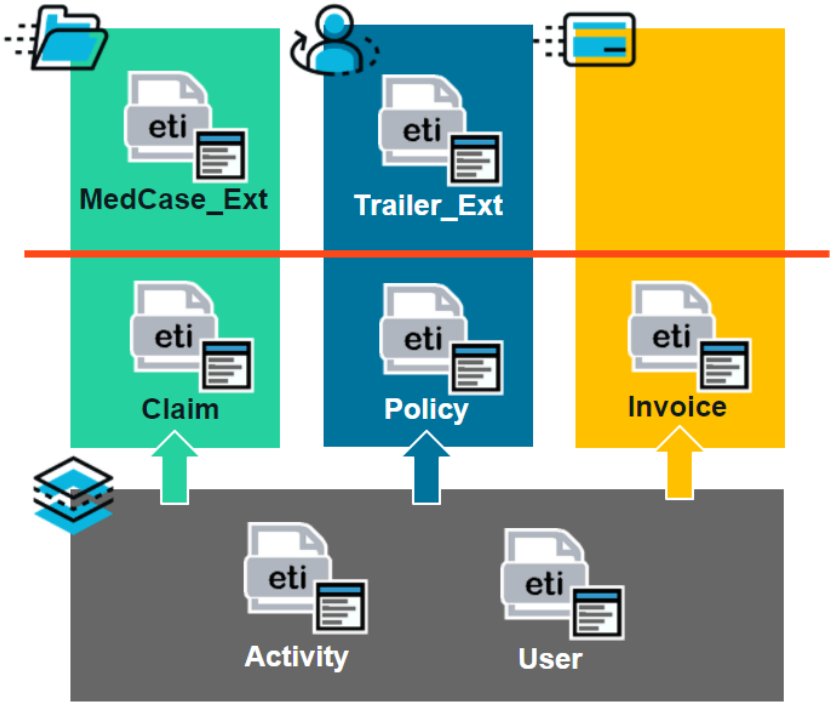
Notes

Review: Entities in the data model (2 of 2)

Custom entities are always found in the … \modules\configuration\config\extensions\entity\ folder.

Menu

Review: Entities in the data model (2 of 2)



- Two main types:
  - Base application entities
  - Custom entities
- A custom entity is an entity created by the customer
- A custom entity can be created only on the application level and it will be local inside that application

Education

Notes

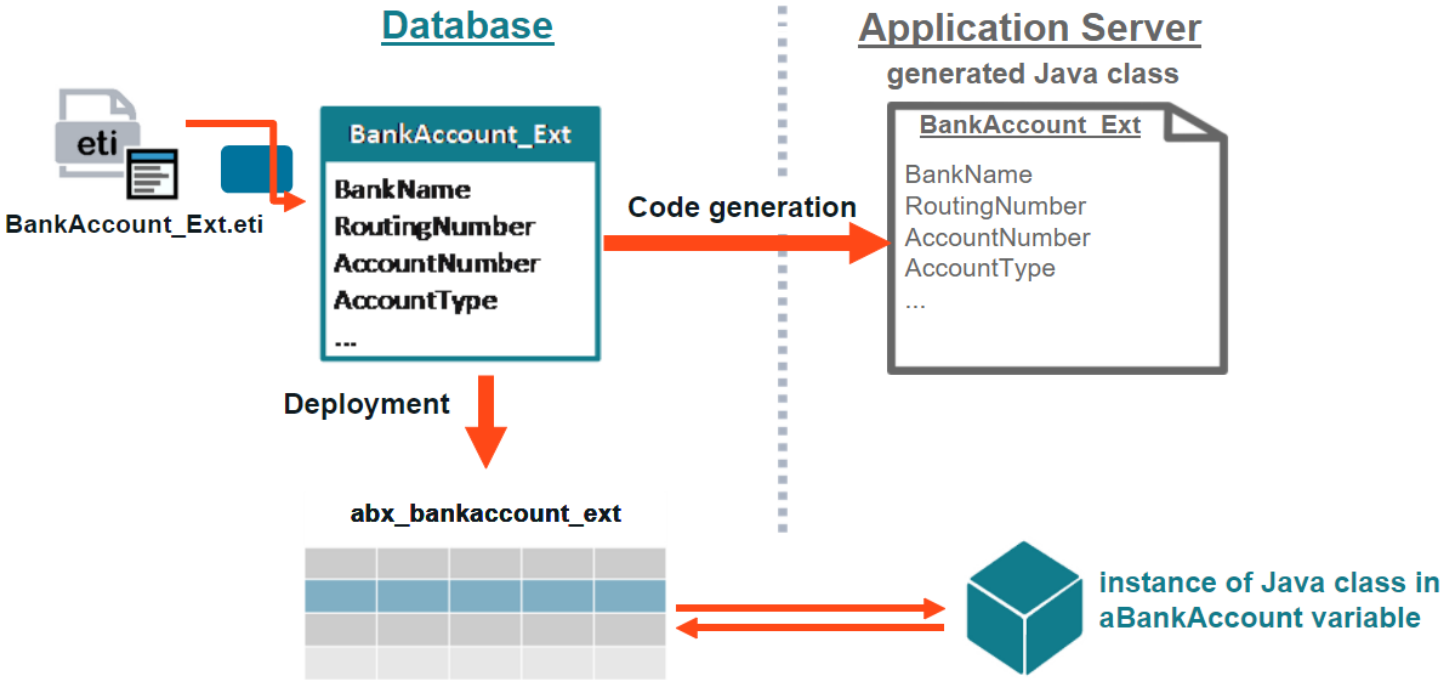
Entities, Java classes and database tables

For a customer data model entity, the ETI file defines the fields. Custom entities are a single ETI file that customers can edit as needed. A customer entity will consist of only an ETI file and it cannot have EIX or ETX files. All the fields in the custom entity's ETI file become a part of the generated Java class and a database table.

For custom entities, Guidewire recommends that the entity name should end with _Ext, for example, BankAccount_Ext. The names of the fields in the custom entity do not need to end with the _Ext suffix.

The name of the database table is automatically generated to match the name of the ETI file and is all lowercase, for example, bankaccount_ext. Guidewire creates a special prefix for custom entities by appending x to the two-letter product code for the table name, such as abx_bankaccount_ext. The table name is an attribute of the entity xml element and must have a length of 25 characters or less. When creating an entity, if the entity name has a length of more than 25 characters, you can modify the table name in the Entity dialog to reduce the length to 25 or less, including removing the _ext suffix from the table name.

Entities, Java classes and database tables



Education

PREV NEXT

Notes

Entities in the database

There are two exceptions to the generalization above: virtual and subtype entities.

Virtual. Non-persistent entities are entities constructed entirely with code. They exist during run-time, and when the Guidewire application shuts down, the data is not saved to the database. An example of a non-persistent entity is ABContactSearchCriteria in TrainingApp. Guidewire recommends that you do not create non-persistent entities as a general rule

Subtype entities are entities that are children entities of a top level supertype (parent) entity. All fields in the parent entity are inherited by the child entity. A top level supertype entity and all of its child subtypes are stored in a single database table. In this manner, all the entities share the same database table.

Entities in the database

```
select bankname, routingnumber, accountnumber, accounttype from abx_bankaccount_ext;
```

BANKNAME	ROUTINGNUMBER	ACCOUNTNUMBER	ACCOUNTTYPE
ACME Credit Union	123-987	345678900	10001
National Bank	745-222	232323566	10001
ACME Credit Union	123-987	112233445	10001
ACME Credit Union	123-987	554433221	10002

(4 rows, 6 ms)

- Often, an entity stores data in a database table named for the entity:
  - BankAccount_Ext entity in abx_bankaccount_ext table
  - ABContact entity in ab_abcontact table
- Exceptions are…
  - Entities that share the columns in an entity table such as a subtype entity
  - Virtual, non-persistent entities do not save data to the database. Guidewire recommends that you do not create non-persistent entities as a general rule

Education

Notes

Data relationships

Now let us take a look at the kinds of relationships that exist between data entities in Guidewire applications.

Menu

Data relationships

Education

PREV NEXT

Notes

Foreign key

If a foreign key field is added to a base application entity, then the suffix should be _Ext.

For a complete reference of the <foreignkey> syntax, refer to the application's Configuration Guide.

Menu

Foreign key

- Pointer to single instance of some other entity
- Example: each Contact can have one assigned user

Education

Notes

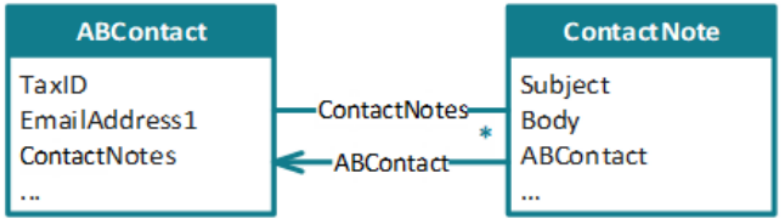
Array

Keep in mind that arrays are maintained in code. The code assembles the array by executing queries against the database. In order to do this, the application must be able to query for all members of a given array. It can do this only if each member has a foreign key referencing its parent. Note that the value of the arrayentity attribute must match exactly the name of the entity it references.

Menu

Array

- Collection of pointers to instances of some other entity that is maintained by code during runtime
- Requires reverse foreign key
  - If entity A has array of entity B, then entity B must have foreign key pointing to entity A
- Example: each ABContact can have zero to many contact notes



Education

PREV NEXT

GUIDEWIRE

Notes

One-to-one relationships

A one-to-one relationship often splits a logical entity across multiple physical entities. For example, when an ABContact is a ABPolicyPerson or ABPolicyCompany, the data model must capture financial summary information.

Storing the financial summary details into the ABContact table would result in a table with numerous rows with null columns. Instead, it is better to have a separate entity manage the financial summary details. In this manner, every ABContact has (at most) one FinancialSummary, and every financial summary applies to (at most) one ABContact.

For more information on one-to-one relationships, refer to documentation.

One-to-one relationships

- <onetoone> element defines a single-valued association to another entity that has a one-to-one cardinality
- Provides a reverse pointer to an entity that is pointing at the <onetoone> entity through a foreign key
- Often splits logical entity across multiple physical entities
- Example: ABContact defines Financial Summary as one-to-one and FinancialSummary defines a foreign key to ABContact

Education

Notes

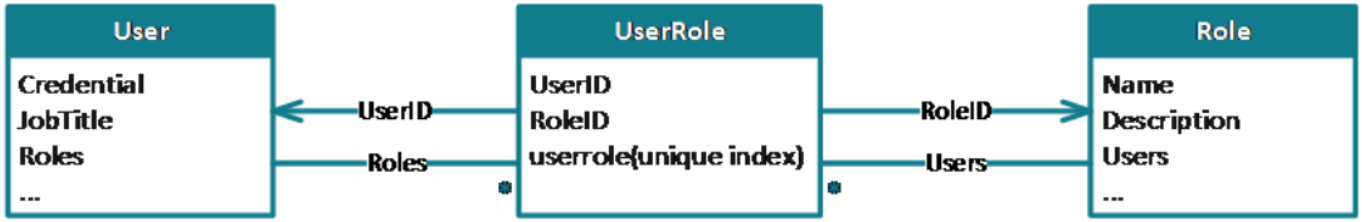
Many-to-many relationships (1 of 2)

To add a many-to-many relationship between entity types to the data model, first create a separate versionable entity that represents the many-to-many relationship. (Versionable entities have a version and ID field and can be deleted as opposed to retireable entities which can never be deleted). Next, create foreign keys to each entity in the many-to-many relationship. Then add a unique index consisting of the foreign keys in the relationship.

Many-to-many relationships require a separate entity that contains the required foreign keys and a unique index for the foreign key values so that duplicate rows are not added to the table.

Many-to-many relationships (1 of 2)

- Requires an entity of type versionable that represents the many-to-many relationship with the following:
  - Two or more non-nullable foreign keys
  - Unique index for all foreign keys



Education

Notes

Many-to-many relationships (2 of 2)

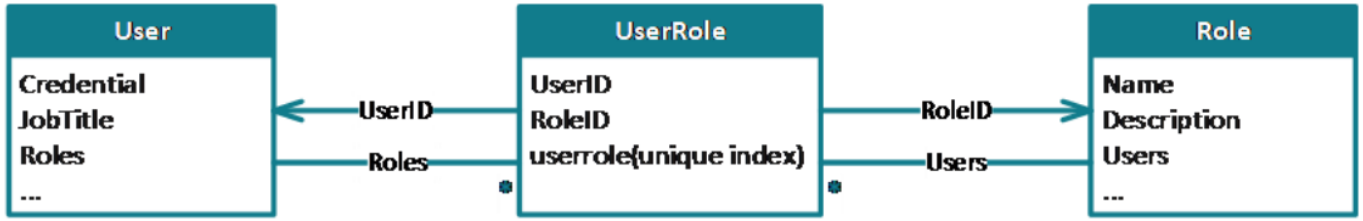
In order to access the many-to-many relationship values, one of the foreign key tables needs to specify an array entity. To access the relationship, add an array to one or both entities in the relationship.

In the slide example, the diagram does not represent the defined relationship in TrainingApp. In TrainingApp, UserRole is of the type joinarray. The joinarray entity type is an internal type and Guidewire does not recommend using this entity type. In addition, only User has a backing Roles array to UserRole.

For more information on many-to-many relationships and entities of type joinarray, refer to the documentation.

Many-to-many relationships (2 of 2)

- To access the many-to-many relationship, add an array to one or both entities in the relationship
  - Roles array in User specifies that there is zero or more UserRoles for a given user
  - Users array in Role specifies that there is zero or more UserRoles for a given role



Notes

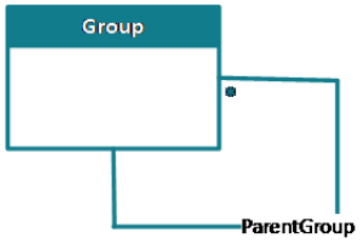
Circular relationships

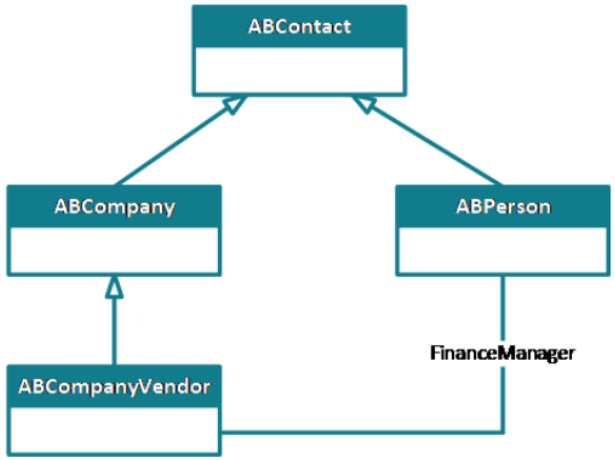
In some cases, an entity needs to refer to itself. This could occur both for a non-subtyped entity (every Group has a parent group) or a subtyped entity when one subtype refers to another (every ABCompany—which is an ABContact—can have a finance manager who is an ABPerson, which is also an ABContact). The Guidewire data model does not let an entity reference itself, or two or more entities reference one another in a cyclical manner, using only foreign keys. This is because Guidewire cannot easily determine the order in which rows can safely be written to the database when a data model cycle exists.

To solve this kind of circular dependency between foreign keys, Guidewire recommends that you use an edge foreign key (<edgeForeignKey>). An edge foreign key from A to B introduces a new, hidden entity that has a foreign key to A and a foreign key to B. However, it does not create any direct foreign key from A to B. As such, ClaimCenter can safely commit the A objects first, then the B objects, and finally, the hidden A/B edge foreign key entities. This ensures that the relationship information does not reference non-existent rows. The same principal is true for two or more entities that reference one another in

Circular relationships

- <edgeForeignKey> creates an edge table
  - Links one entity to another or to self
  - Ensures the safe ordering of data insertion and deletion in cases where cyclic references prevent safe ordering
- Examples:
  - Child references parent
  - Reference between subtypes





Education

Notes

How many ETX files can a given entity have?

Menu

Answer

How many ETX files can a given entity have?

One ETX file per entity in most cases.

Education

PREV NEXT

Notes

Describe some of the differences between the files in the ... \Metadata\Typelist\ and ... \Extensions\Typelist\ folders.

Menu

Answer

Describe some of the differences between the files in the …\Metadata\Typelist\ and …\Extensions\Typelist\ folders.

Files under Metadata are read-only and are specific to the base configuration. You cannot create Typelist files in the Metadata directory but can create Typelist files in the Extensions directory. Only internal Typelist extension files (TIX) are in the Metadata directory. Only external Typelist extension files (TTX) are in the Extensions directory. Both directories contain Typelist (TTI) files.

Education

PREV NEXT

GUIDEWIRE

Notes

User Story: Invoicing Method

Shown here is an example user story requirement requested by a fictional business analyst.

Menu

User Story: Invoicing Method

"After further analysis, we decided that the invoicing method of each policy also has to be stored in TrainingApp. The invoicing method specifies how the premium will be collected for the policy. An insurance company allows only the following 3 invoicing methods:

- *10% down payment and 11 installments*
- *30% down payment and 4 installments*
- *50% down payment and 2 installments*

We are still working on the UI requirements, so please configure the Data Model only for now."

- Insurance Company Business Analysts

Education

Notes

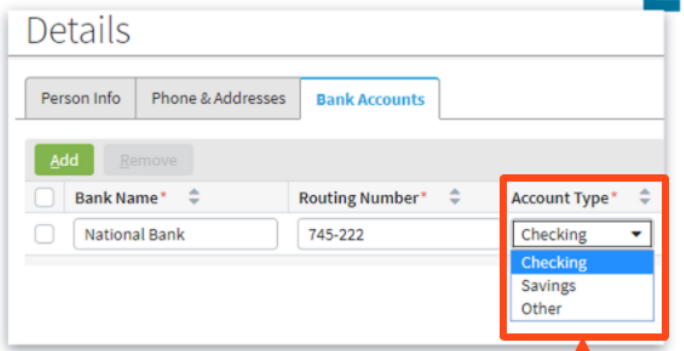
Typelist

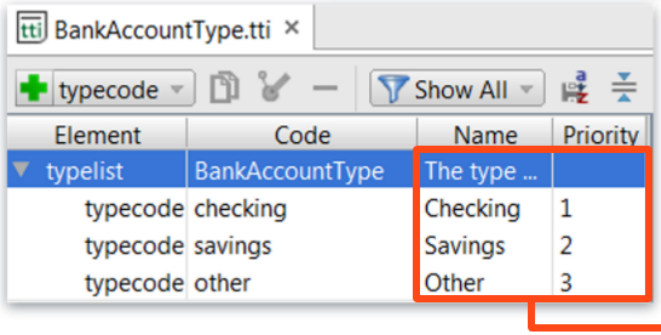
Whenever there is a drop-down list in an InsuranceSuite application interface, it is usually a Typelist. If Guidewire defines a typelist as final, it is not possible to add or delete typecodes from the typelist.

Menu

Typelist

- A Typelist is a fixed, predefined list of values (Typecodes)
  - Values cannot be dynamically added or removed during runtime
- Defined value often used to constrain user input
  - When rendered in UI, typically appears as a dropdown list





Education

Notes

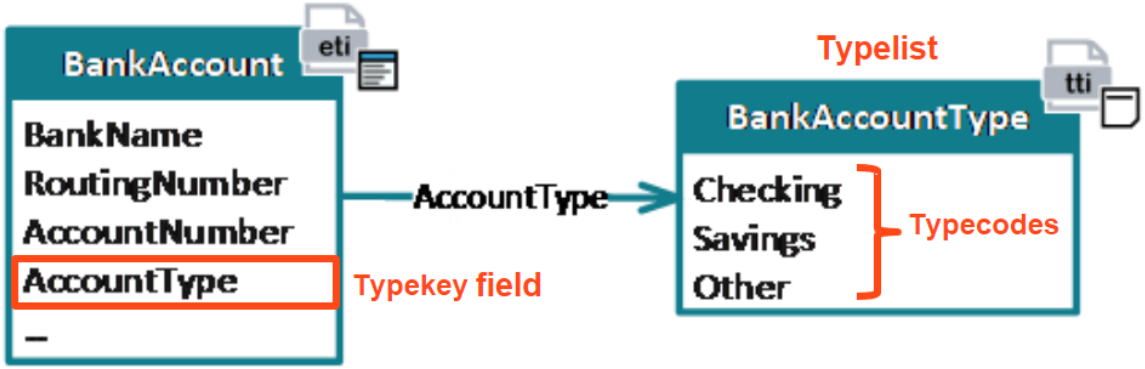
Typelists and Typekey fields

The specified typelist limits the available field values to those defined in the typelist. If you filter the typelist, the field displays a subset of the typelist values.

Menu

Typelists and Typekey fields

- A Typekey field is an entity field associated with a specific Typelist
- Referenced Typelist contains the list of possible values for the Typekey field
  - The Typekey field can point to one of these Typecodes



Education

PREV NEXT

GUIDEWIRE

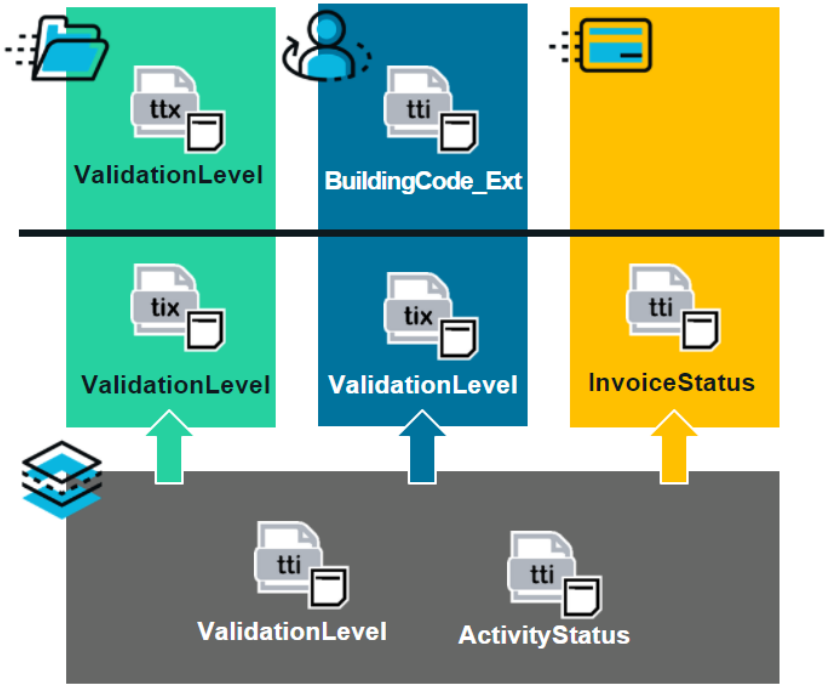
Notes

Typelists in the data model

For an entity to be able to access and use a typelist, you need to define a <typekey> element on that entity. Use the <typekey> element to specify the typelist in the entity metadata.

Menu

Typelists in the data model



- Follow similar principles as entities
  - Defined as XML files
  - TTI, TIX and TTX files
  - …\Metadata\Typelist folder read-only and contains platform and base application Typelists
  - …\Extensions\Typelist folder contains everything customer specific

Education

PREV NEXT

GUIDEWIRE

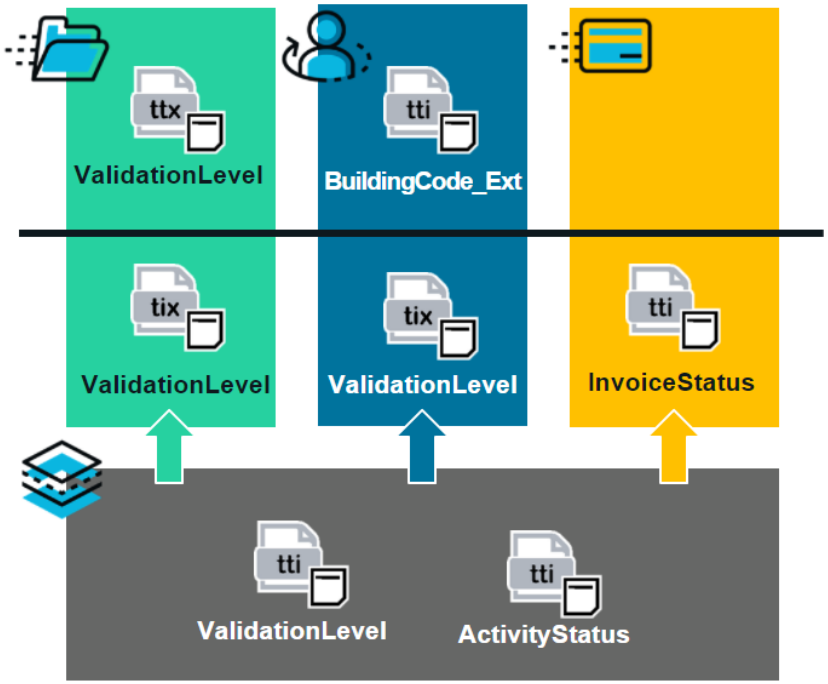
Notes

Typelists in the data model (Continued)

For an entity to be able to access and use a typelist, you need to define a <typekey> element on that entity. Use the <typekey> element to specify the typelist in the entity metadata.

Menu

Typelists in the data model (Continued)



- Kinds of Typelists:
  - ValidationLevel is extendable
  - ActivityStatus is internal
  - BuildingCode_Ext is custom
- Virtual typelists don't physically exist in the data model

Education

PREV NEXT

Notes

Typelists and Typekey fields in the database

Making changes to Typelists, Typecodes, and Typekeys generate changes to the Data Model. Guidewire recommends that you regenerate the InsuranceSuite application's Data Dictionary if you make any updates to the Data Model. If you have made any mistakes in the previous steps, regenerating the data dictionary helps to identify those mistakes.

Menu

Typelists and Typekey fields in the database

- Typelists are stored as database tables
- Typecodes are stored as rows in the table
- Typekeys are stored as foreign key columns in the entity table

ID	TYPECODE	NAME	RETIRED
10001	checking	Checking	FALSE
10002	savings	Savings	FALSE
10003	other	Other	FALSE

BANKNAME	ROUTINGNUMBER	ACCOUNTNUMBER	ACCOUNTTYPE
ACME Credit Union	123-987	345678900	10001
National Bank	745-222	232323566	10001
ACME Credit Union	123-999	112233445	10001
ACME Credit Union	225-999	554433221	10002

Education

Notes

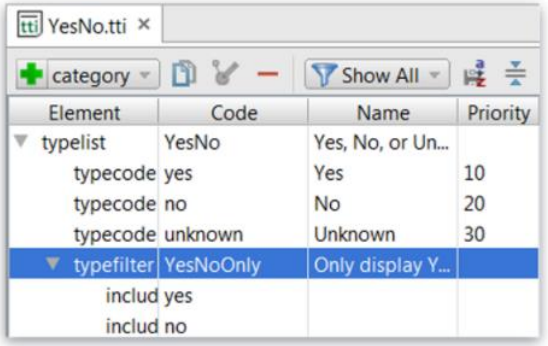
Typefilters

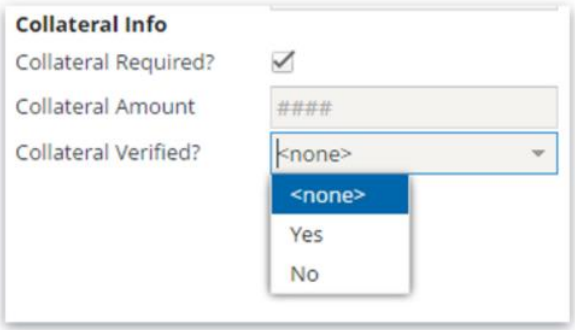
A Typefilter defines a named subset of the Typecodes that could be used as value for the associated Typekey field that references this Typelist.

Menu

Typefilters

- A Typelist filter causes the Typelist to display only a subset of the Typecodes for that Typelist
- Multiple Typefilters can be defined in a Typelist
- A Typekey field can reference at most one Typefilter from the associated Typelist





Education

PREV NEXT

GUIDEWIRE

Notes

What are some of the differences between the files in the ...\Metadata\Typelist\ and ...\Extensions\Typelist\ folders? (Answer)

Menu

Answer

What are some of the differences between the files in the ...\Metadata\Typelist\ and ...\Extensions\Typelist\ folders?

- Files under Metadata are read-only and are specific to the base configuration.
- You cannot create Typelist files in the Metadata directory but can create Typelist files in the Extensions directory.
- Only internal Typelist extension files (TIX) are in the Metadata directory.
- Only external Typelist files (TTX) are in the Extensions directory.
- Both directories contain Typelist (TTI) files.

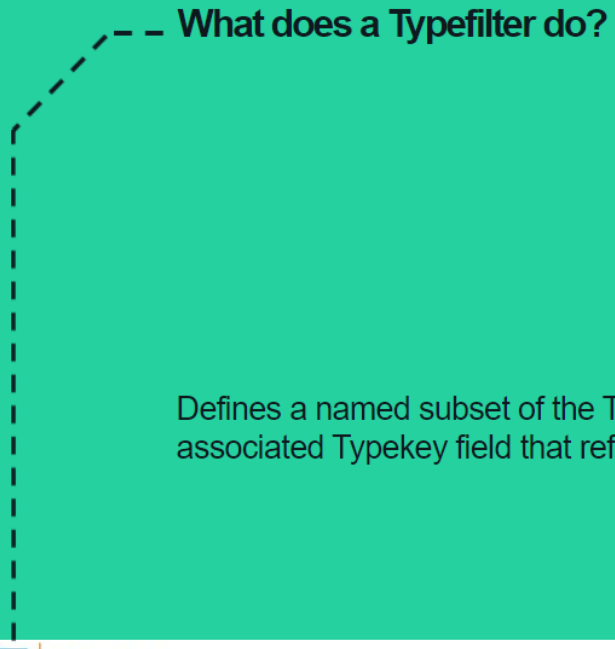
Education

Notes

What does a Typefilter do? (Answer)

Menu

Answer



What does a Typefilter do?

Defines a named subset of the Typecodes that could be used as value for the associated Typekey field that references this Typelist.

Education

PREV NEXT

GUIDEWIRE

Notes

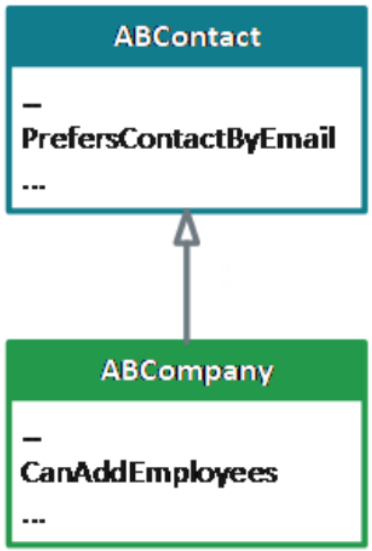
Subtype entity

Note: Language primitives also have types. For example, the code "typeof 29" is valid Gosu, and it returns java.lang.Integer, which is a Type. In other words, Integer is a subtype of Type.

Menu

Subtype entity

- A subtype is an entity that is a child to a supertype entity and inherits all fields and methods from the supertype
- Example:
  - ABContact is the supertype
  - ABCompany is the subtype
  - ABCompany inherits PrefersContactByEmail from ABContact
  - ABContact does **NOT** inherit CanAddEmployees



Education

PREV NEXT

GUIDEWIRE

Notes

Example: subtype hierarchy in TrainingApp

There is one subtype entity excluded for the diagram because it is not directly relevant to the configuration lab work: ABUserContact. ABUserContact represents a TrainingApp user and defines a user's contact details.

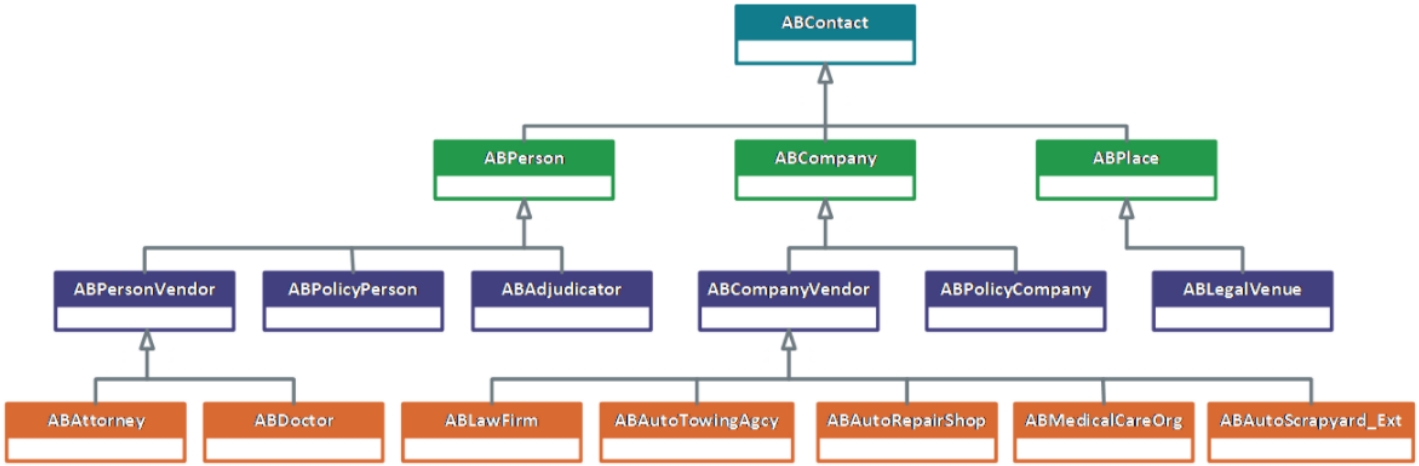
ABContacts are organized into a set of subtypes. The organization of the hierarchy helps to model information about contacts. Information common to all contacts can be established at the ABContact level. It is automatically inherited by all of its subtypes. Information specific to an ABPerson can be defined at the ABPerson level. It is inherited by all of its subtypes, but information on the ABPerson subtype is not available to ABCompany or ABPlace.

In the ABContact hierarchy, an ABPolicyPerson is a person who owns a policy issued by the carrier (such as an individual with a personal auto policy). An ABPolicyCompany is a company which owns a policy issued by the carrier (such as a construction company with a workers' compensation policy).

Subtyping is also used within the three primary applications. For example:

- BillingCenter's Activity, Plan, and Contact entities

Example: subtype hierarchy in TrainingApp



Education

PREV NEXT

Notes

Additional subtype hierarchy examples

In ClaimCenter:

- A transaction is used to denote the movement of money either into a reserve line or from a reserve line to a payment.
- An incident describes a damaged or lost property or coverable.

In PolicyCenter:

- A job is used to manage a policy transaction, which either creates or modifies a policy.
- A plan detail is used to manage plans, which typically identify how the insured will be billed for or will pay for the policy. This information is passed to the billing system.

In BillingCenter:

- A plan is used to store information about how Invoicing, Commissions, Billing and Delinquency are performed.
- A charge pattern is used to define billing behaviors for each kind of charge, such as installment invoicing for Premium charges versus one time billing for Taxes.

Additional subtype hierarchy examples

All applications

Contact: Person, Company, Place, ...

ClaimCenter

Incident: InjuryIncident, TripIncident, PropertyIncident

Transaction: Payment, Recovery, Reserve, ...

PolicyCenter

Job: Submission, Renewal, Cancellation, Reinstatement, ...

PlanDetail: BillingPlanDetail, PaymentPlanDetail, ...

BillingCenter

Plan: BillingPlan, CommissionPlan, DelinquencyPlan, ...

ChargePattern: ImmediateCharge, ProRataCharge, ...

Education

Notes

Subtypes in the database

ABContact is the supertype for the three primary subtypes : ABPerson, ABCompany, and ABPlace. There are a total of 17 subtypes of ABContact for TrainingApp.

For columns not relevant to a given row's subtype, the column contains a null value. For example, Lily Watson is an ABAttorney and therefore the Name column is null value. Similarly, Express Auto, an ABAutoRepairShop, shows null values for FirstName and LastName. TaxID is a field defined in the ABContact entity. Both Express Auto and Lily Watson have TaxID encrypted values.

High degrees of subtyping of a single supertype can result in a highly denormalized database table with many columns and large row sizes. For purists seeking high degrees of normalization in database tables, subtyping will appear abhorrent.

In Guidewire applications, subtypes represent a trade-off between data model configuration flexibility and database performance. High degrees of normalization require numerous joins when querying. In many ways, it can be easier to add columns to a table rather than create more table objects in a database that require numerous joins for queries.

Subtypes in the database

- Typelist and entity table define supertype and subtype relationships
- Supertype entity table contains subtype columns in denormalized form
  - **ID**, **TaxID** and **Name** from ABContact
  - **FirstName** and **LastName** from ABPerson
  - **AttorneyLicense** from ABAttorney
  - **IsFranchise** is from ABAutoRepairShop
- Irrelevant columns for subtype are null

SELECT ID, TYPECODE
FROM ABTL_ABCONTACT
ORDER BY ID;

ID	TYPECODE
1	ABAdjudicator
2	ABAttorney
3	ABAutoRepairShop
4	ABAutoScrapYard_Ext
5	ABAutoTowingAgcy
6	ABCompany
7	ABContact
8	ABDoctor
9	ABLawFirm
10	ABLegalVenue
11	ABMedicalCareOrg
12	ABPerson
13	ABPlace
14	ABPolicyCompany
15	ABPolicyPerson
16	ABUserContact
17	ABCompanyVendor
18	ABPersonVendor

(18 rows, 4 ms)

ID	TAXID	NAME	FIRSTNAME	LASTNAME	ATTORNEYLICENSE	ISFRANCHISE	SUBTYPE
67	2096542-3113-2456902	*null*	Lily	Watson	J12-13562	*null*	2
68	9983200-5335-0023899	*null*	James	Andersen	S20-82325	*null*	2
75	1242577-7777-7752421	Express Auto	*null*	*null*	*null*	*null*	3
76	3219251-8888-1529123	European Autoworks	*null*	*null*	*null*	*null*	3

Education

PREV NEXT

Notes

Review: subtypes

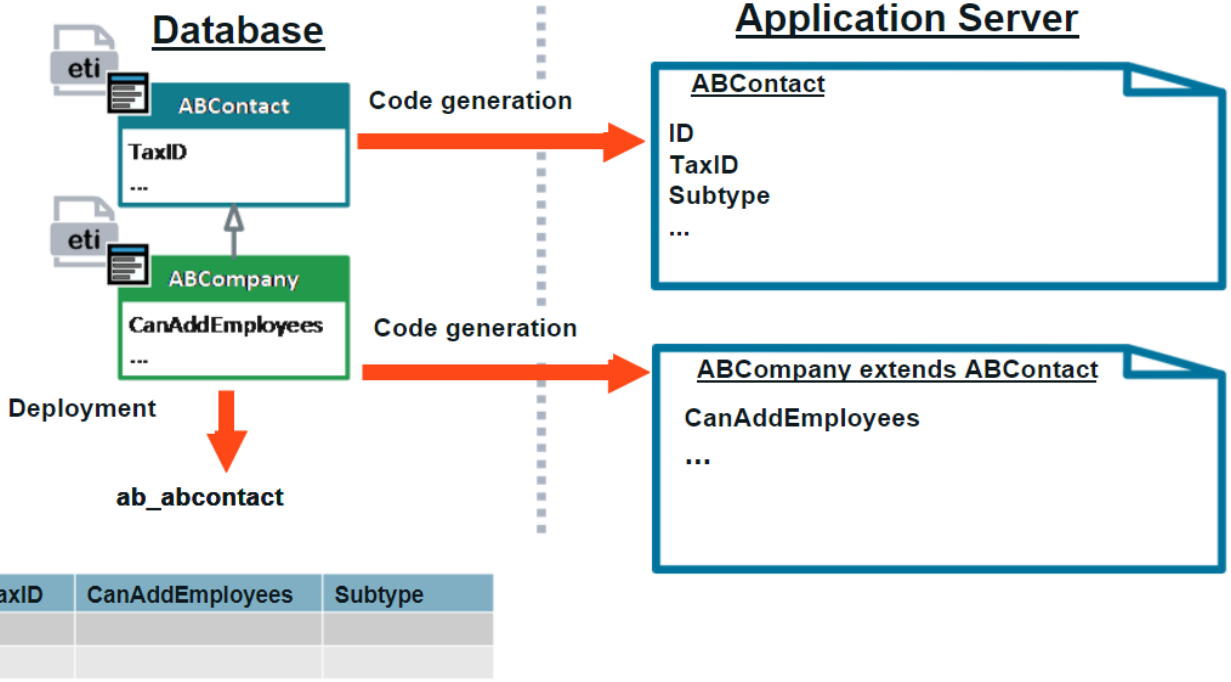
In object-oriented design, a class can be extended by a child class or subclass. Every field and method declared at the parent class is added to the child class automatically. In objected-oriented design, subclass extensions from a superclass is called inheritance.

Guidewire subtypes also make use of inheritance. Every subtype has its own generated Java class. This class inherits all fields and methods of the parent class.

In most cases, a database table corresponds to one data model entity, and therefore one database-backed Gosu class. One exception to this is where subtypes are concerned. If an entity is subtyped, then the database table corresponds to the top-level data model entity and all of its subtype entities. The table then also corresponds to one top-level database-backed Java class and all of its subclasses.

For example, the ab_abcontact table has column for CanAddEmployees. This field only applies to ABContacts that are of type ABCompany. For ABPersons, that column will have null value.

Review: subtypes



Education

PREV NEXT

Notes

What is a subtype entity? (Answer)

Menu

Answer

What is a subtype entity?

A subtype is an entity that is a child to a parent entity and inherits all fields from the parent.

Education

PREV NEXT

Notes

What is a subtype entity extension? (Answer)

Menu

Answer

What is a subtype entity extension?

A subtype extension is an entity extension of a base application subtype entity.

Education

PREV NEXT

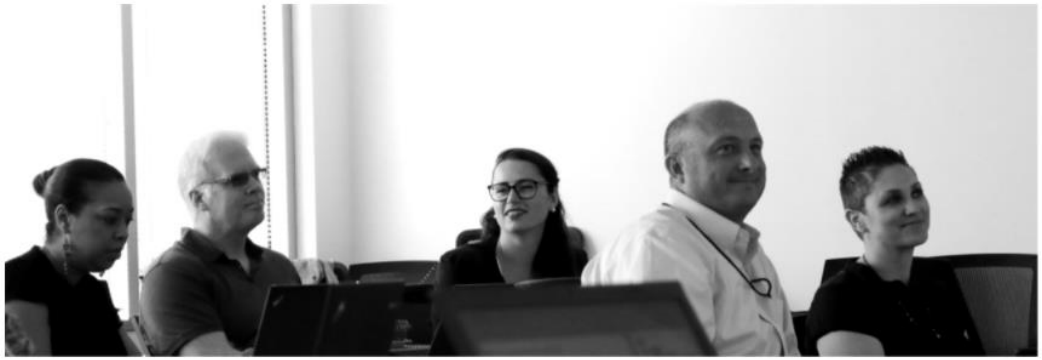
Notes

The User Interface Architecture

Menu

Education

The User Interface Architecture



NEXT

Notes

The Guidewire UI Architecture

Menu

The Guidewire UI Architecture

Education

PREV NEXT

Notes

Guidewire application UI framework

A PCF is an XSD-validated XML document containing elements that describe the structure, layout, and behavior of the web user interface. The majority of this content is defined at the application level, though some is defined at the platform level.

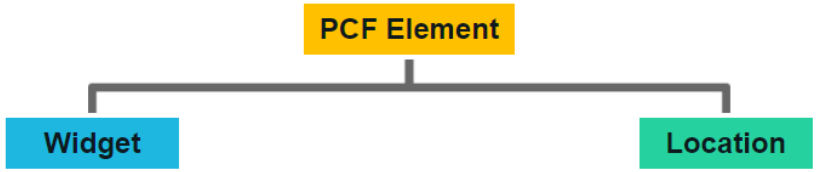
A widget is a PCF element that will be converted into HTML and displayed. This does not mean that a widget itself will always be visible. Some of the widgets are containers that are only used to group other widgets in a specific layout format. Widgets can specify permissions that the logged in user needs to view/edit the data.

A location is a PCF element that a user (or the application itself) can navigate to. It is used to define how users move from one area of the user interface to the next. A location usually specifies:

1) which part of the browser window will be updated if the application/user navigates to the location
2) what are the permissions that the logged in user needs to view/edit the data in that location

Note: Both Widget and Location are conceptual representations in this diagram. There are no <Widget /> or <Location /> elements.

Guidewire application UI framework



Displayable elements of the user interface rendered into HTML

Navigable places in the application that a user or the application itself can navigate to

- The PCF (Page Configuration Format) object model is a proprietary application framework used to create all Guidewire end-user interfaces
- By using the Studio Page Configuration (PCF) editor, you can modify an existing PCF file or add a new PCF file and graphically build and manage its elements

Education

Notes

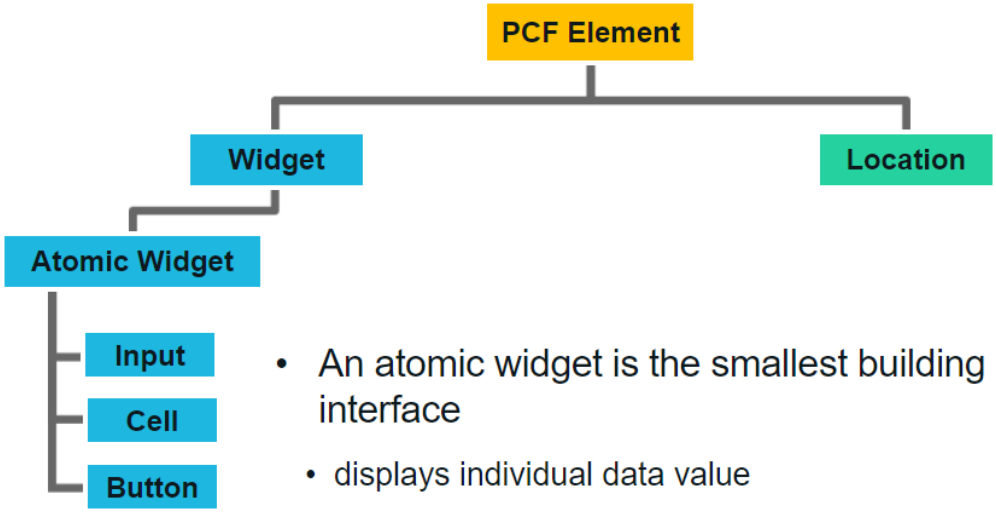
Atomic widgets

Atomic widgets are individual field items such as inputs, cells, or buttons. They will always be defined within a container widget or location.

Note: Both Widget and Location are conceptual representations in this diagram. There are no <Widget /> or <Location /> elements. Atomic Widget is also a conceptual representation. There is no <Atomic Widget /> element.

Menu

Atomic widgets



- An atomic widget is the smallest building block of the user interface
  - displays individual data value
  - executes individual action

Education

PREV NEXT

Notes

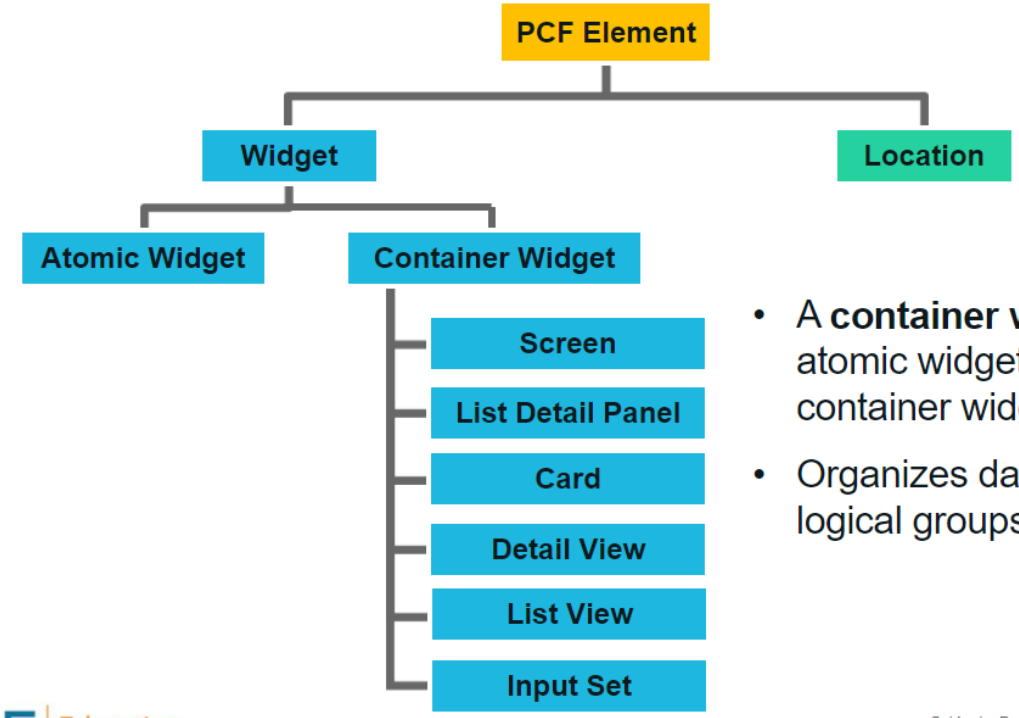
Container widgets

Container widgets hold other widgets. Each one can be defined either in its own file or as a child container within some other PCF element file.

Note: Both Widget and Location are conceptual representations in this diagram. There are no <Widget /> or <Location /> elements. Similarly, both Atomic Widget and Container Widget are conceptual representations. There are no <Atomic Widget /> or <Container Widget /> elements.

The PCF object model is container-based. Each UI element is modeled as an object, which may contain other objects. The hierarchical structure simplifies the task of locating and modifying visual elements. Furthermore, each element can be declared as an independent and therefore reusable element.

Container widgets



- A **container widget** is a collection of atomic widgets and/or other container widgets
- Organizes data and functionality into logical groups

Education

PREV NEXT

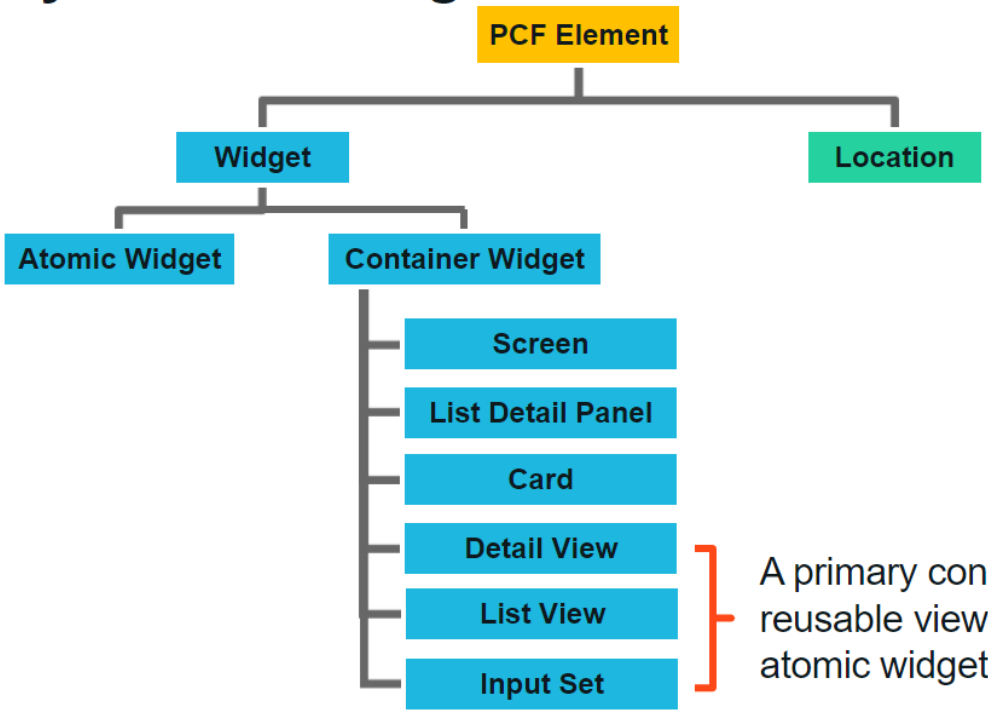
Notes

Primary container widgets

Detail View Panels, List View Panels, and Input Sets are Primary container Widgets. Secondary container widgets are discussed further in this presentation.

Menu

Primary container widgets



A primary container is a reusable view that organizes atomic widgets

Education

Notes

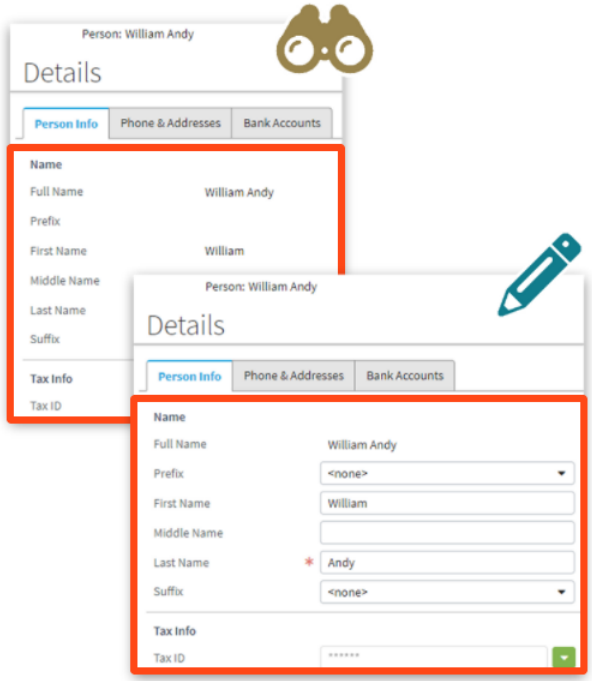
Detail View Panels

Detail view panels are generally designed to use widgets to organize and manage a collection of related pieces of information that constitute a single record. The most obvious example is the set of individual fields of an instance of an entity such as User, Claim, or Policy. There are no constraints imposed by the system as to what information can be displayed by a detail view, however; the view could present information from multiple entities that do not even have to be related to one another, in addition to information that does not derive from an entity at all (such as calculated or static values).

The purpose of a detail view panel, however, and the recommended practice behind their use, is to present the user with all of the detailed information that is relevant to that interface, and no information that is irrelevant. In general, this means displaying information about the fields of a particular entity and possibly fields from a small number of closely related entities.

Detail view panels must contain at least one column and can contain as many columns as necessary to present the widgets in the most effective way for the user.

Detail View Panels



- Focus on a single record
  - Typically, but can be more than one
- Allow the user to…
  - View an existing record
  - Create a new record
  - Edit a record
- Columns help organize atomic widgets to display a related information
- Screens and secondary views can reference
- Often referred to as "Detail View"

Education

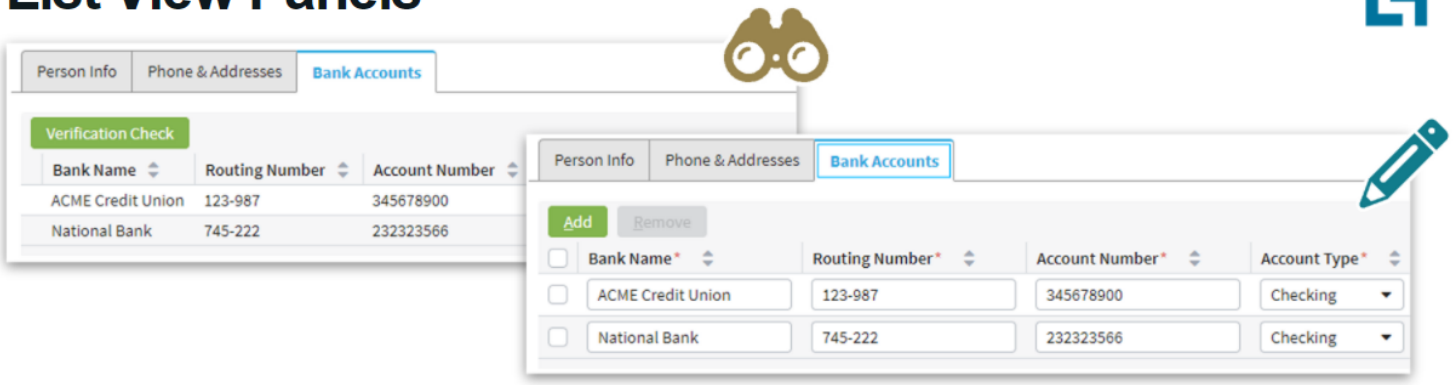
Notes

List View Panels

Just as a detail view panel is designed to allow the user to view detailed information about a single record, a list view panel is designed to allow the user to view summary information about a collection of records.

Menu

List View Panels



- Table organizes information that allows the user to view information about multiple records at the same time
- Uses a small number of atomic widgets to display the most relevant information
- Screens and secondary views can reference
- Often referred to as "List View"

Education

PREV NEXT

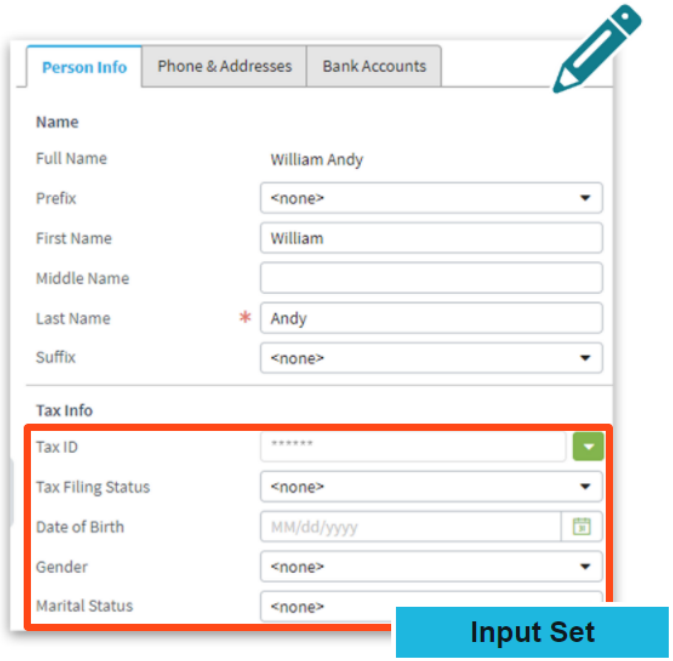
Notes

Input Sets

For more information on Detail View Panels, see the product documentation.

Menu

Input Sets



- Set of atomic widgets grouped together
  - Could be reused together
  - Common visibility / editability condition could be applied across multiple widgets
  - The input set could be targeted with a partial page update to improve UI performance
- Input Sets fit some attributes of the primary containers
  - They can contain atomic widgets and organize them into logical groups
- BUT:
  - Input Sets must be embedded into Detail View Panels
  - Cannot be referenced by secondary views
  - Cannot have a toolbar directly associated with them

Education

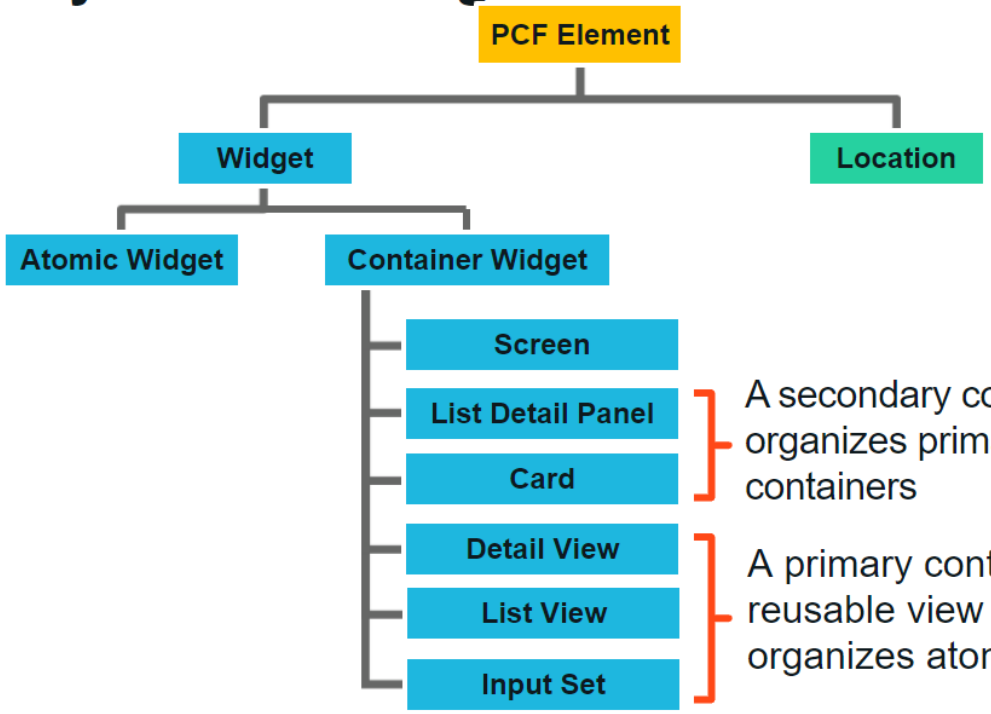
Notes

Secondary container widgets

Primary containers are discussed in earlier slides in this presentation.

Menu

Secondary container widgets



Education

Notes

Secondary containers

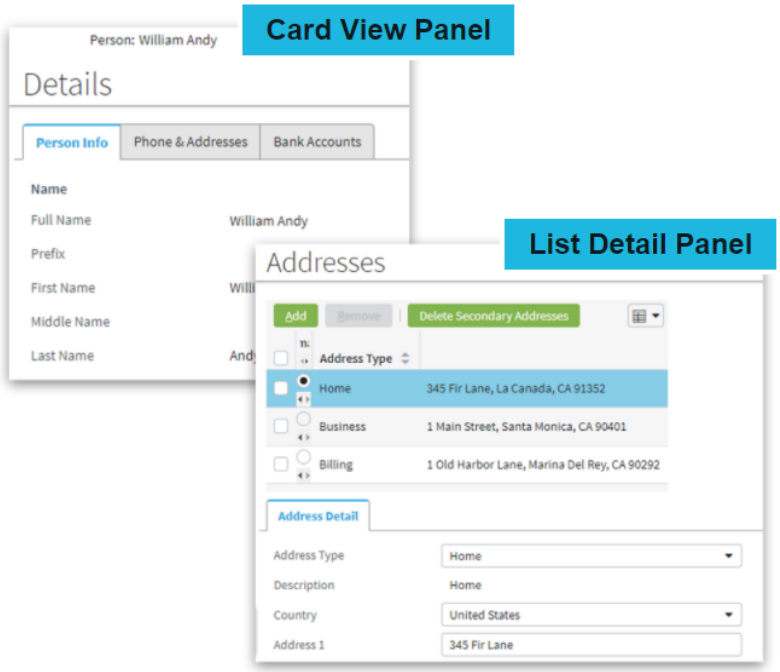
A secondary container organizes primary containers.

Menu

Secondary containers

- A **Card** is a collection of cards, with each card containing Detail Views or List Views

- A **List Detail Panel** contains a top List View and a bottom view panel that display data about the selected list item



Education

PREV NEXT

GUIDEWIRE

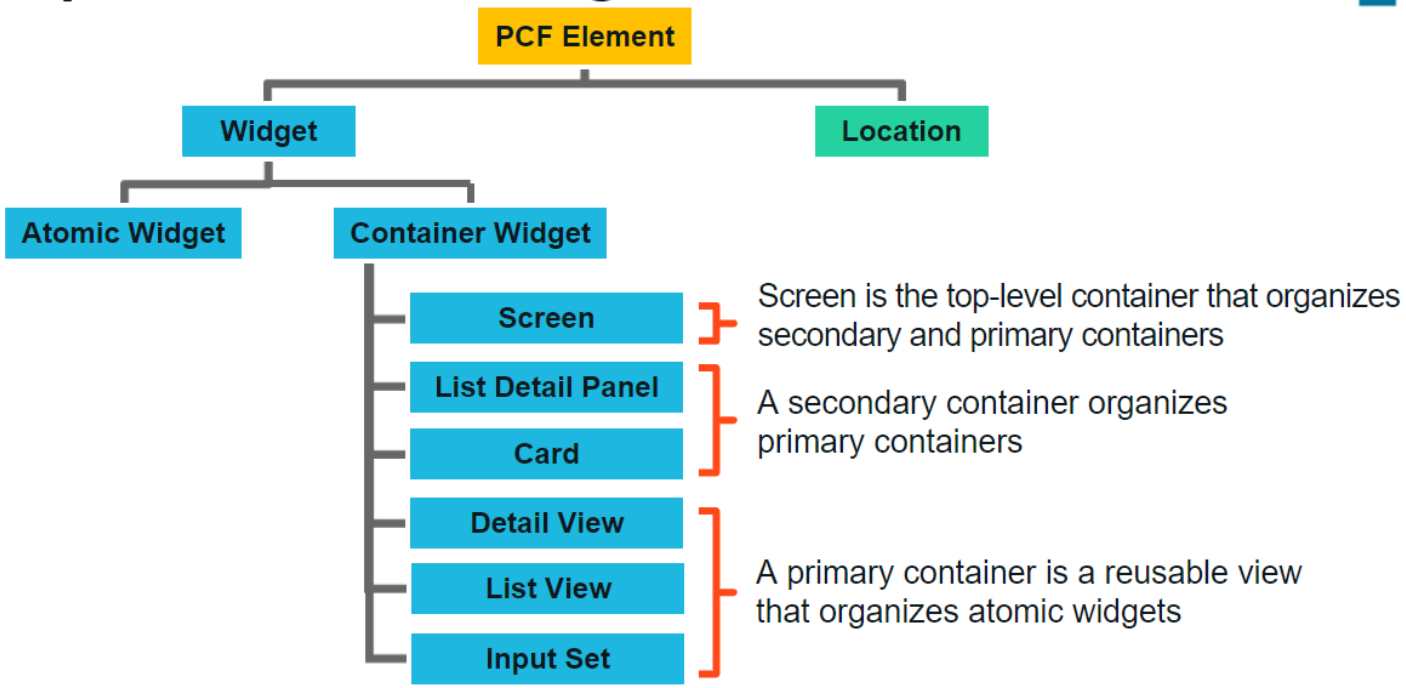
Notes

Top-level container widget

Top level containers organize secondary and primary containers.

Menu

Top-level container widget



Education

PREV NEXT

GUIDEWIRE

Notes

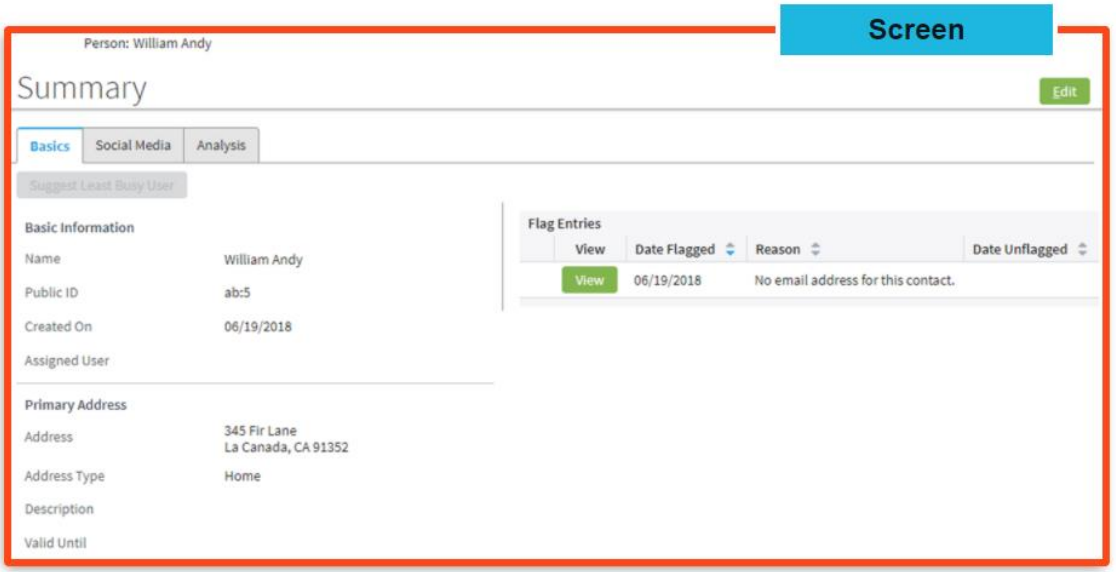
Screen

Shown here is an example of a Screen.

Menu

Screen

Every atomic widget, primary view, and secondary view is contained (directly or indirectly) in a screen.



Education

PREV NEXT

GUIDEWIRE

Notes

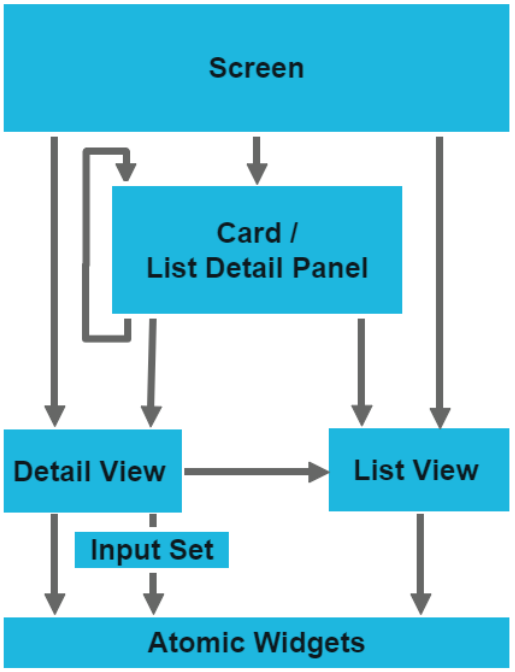
Review: Container widget hierarchy

The arrow indicates that A can contain B. For example, a List View Panel can contain Atomic Widgets but cannot contain a Detail View Panel.

Container widgets can be classified in four categories:

1. The lowest level is comprised of atomic widgets.
2. Atomic widgets can be directly contained only by primary views (detail view panels and list view panels). The purpose of a primary view is to organize atomic widgets into logical groups.
3. Secondary views (card view panels and list detail panels) organize primary views. Secondary views cannot directly contain most atomic widgets.
4. Screens are the top-level containers. Screens are used to connect what is displayed in the user interface with how users navigate through the user interface. A screen can directly contain both primary views and secondary views, but it cannot directly contain most atomic widgets.
5. There is one exceptional case, when a primary view can be placed into another primary view. This will be discussed

Review: Container widget hierarchy



- Screen
  - Top-level container that organizes secondary and primary containers
  - Referenced by locations: Page, Wizard, Popup, Worksheet
- Secondary containers
  - Collections of primary containers organized for usability
  - Could also contain other secondary containers
- Primary containers
  - A reusable view that organizes atomic widgets
  - In certain cases, a Detail View could also contain a List View Panel
- Atomic widgets
  - Individual elements of data and/or functionality

Education

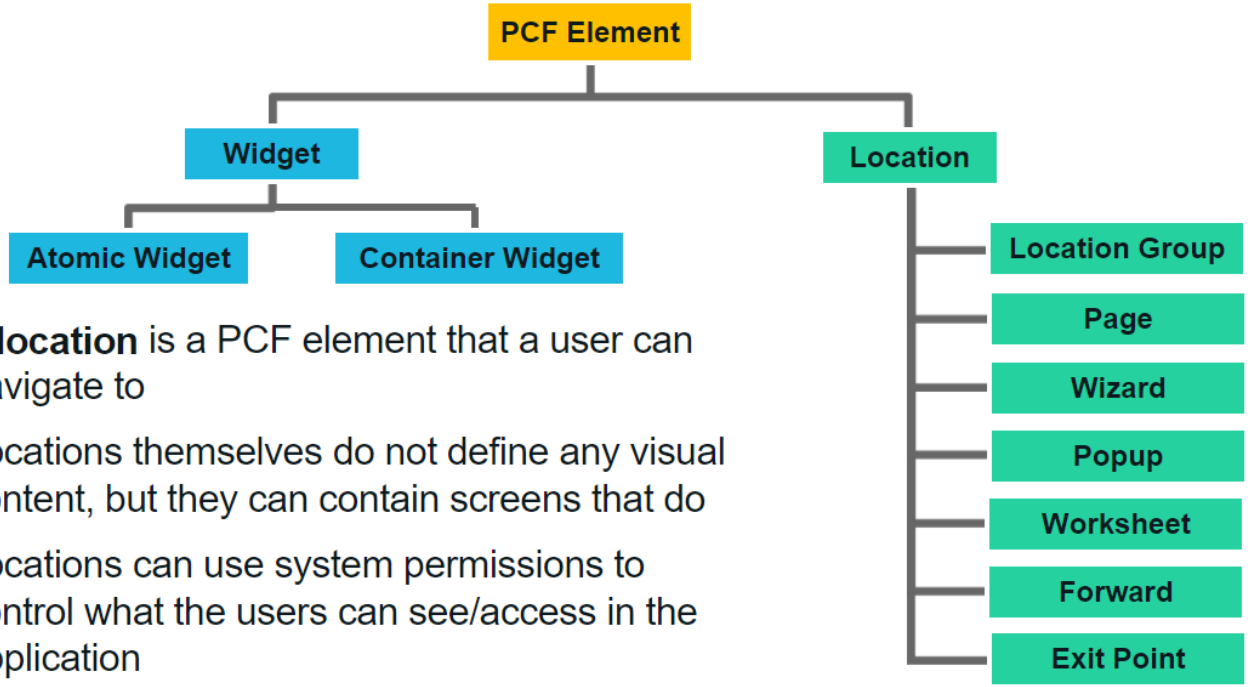
Notes

Locations

Note: Both Widget and Location are conceptual representations in this diagram. There are no <Widget /> or <Location /> elements. Similarly, both Atomic Widget and Container Widget are conceptual representations. There are no <Atomic Widget /> or <Container Widget /> elements.

Menu

Locations



- A **location** is a PCF element that a user can navigate to
- Locations themselves do not define any visual content, but they can contain screens that do
- Locations can use system permissions to control what the users can see/access in the application

Education

PREV NEXT

GUIDEWIRE

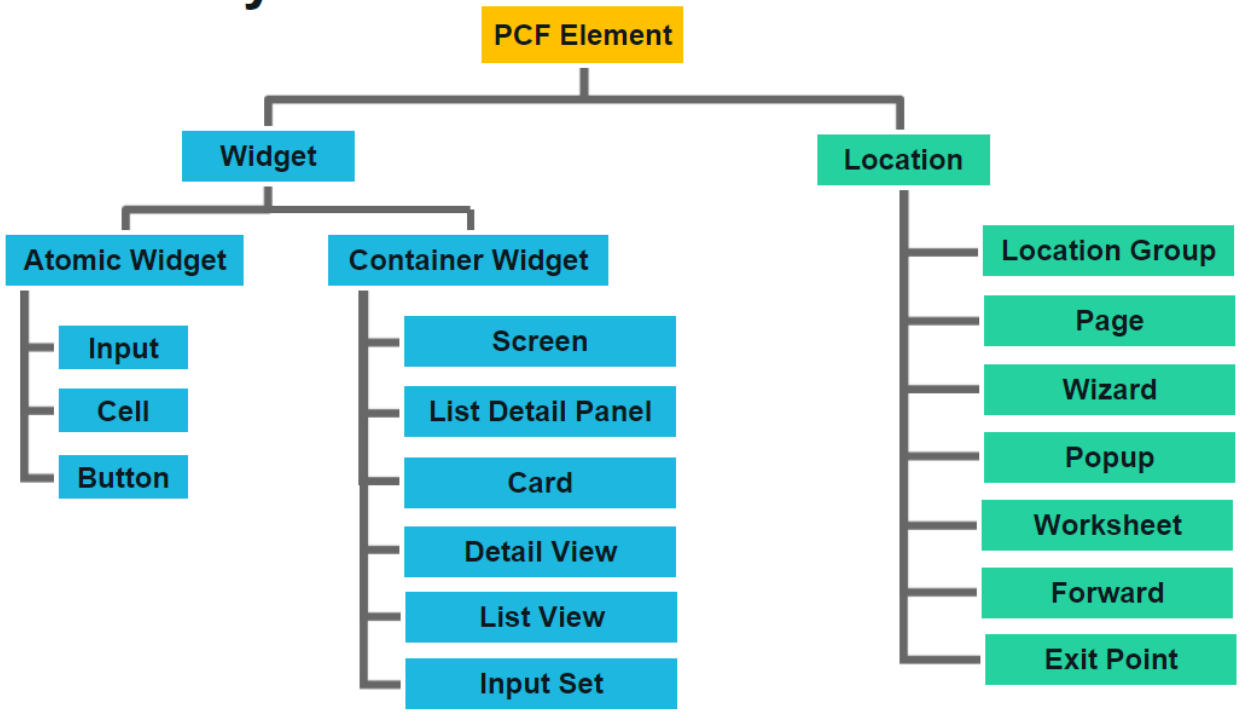
Notes

PCF hierarchy review

At the top level, PCF elements are split between widgets and locations. This lesson only addresses the high-level classifications of widgets (both atomic and container) and locations. Discussions of specific widgets and locations will follow in future lessons.

Menu

PCF hierarchy review



Education

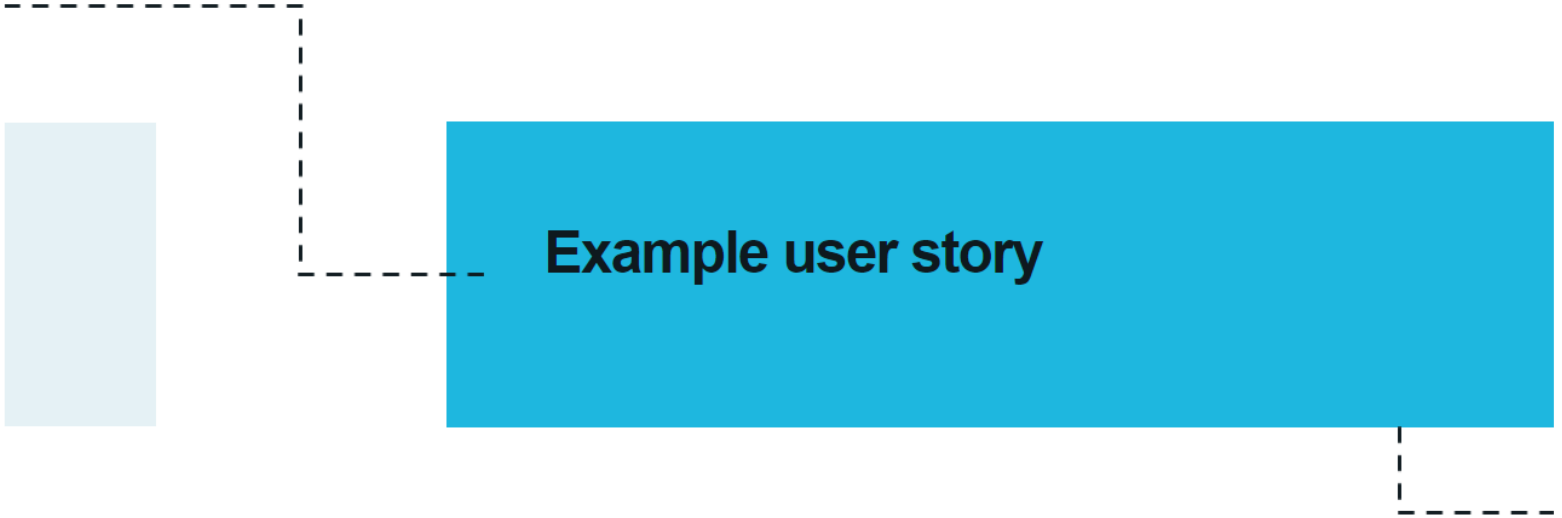
PREV NEXT

GUIDEWIRE

Notes

Example user story

Menu



Education

PREV NEXT

Notes

Define the broad purpose for each element category:

Menu

Answer

Define the broad purpose for each element category:

A) Atomic widgets

B) Container widgets

C) Locations

The correct responses are:

A) Atomic widgets display individual elements of data and/or functionality.

B) Container widgets group atomic widgets into logical groups.

C) Locations define how users move from one place in the application to the next.

Education

PREV NEXT

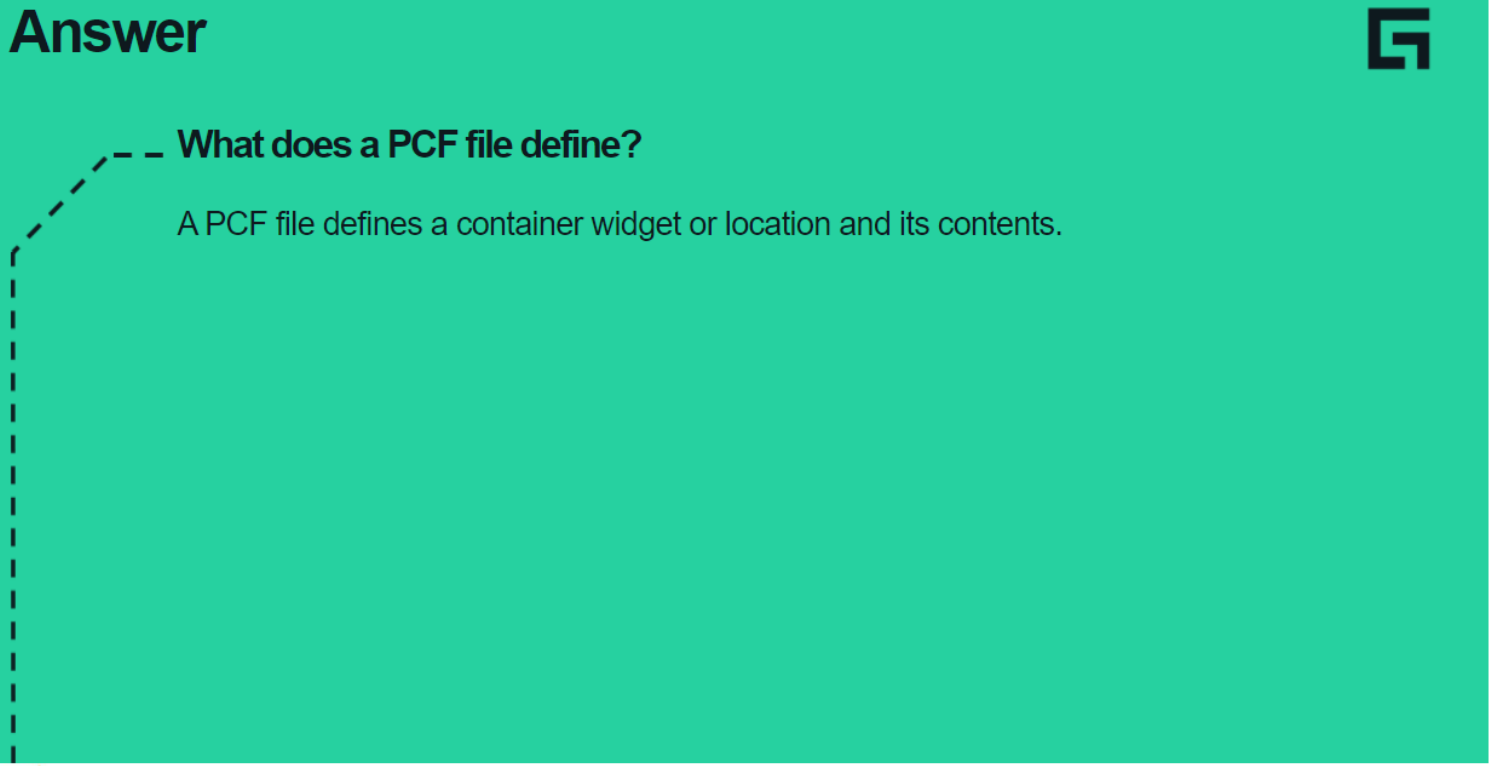
GUIDEWIRE

Notes

What does a PCF file define?

Menu

Answer



Education

PREV NEXT

GUIDEWIRE

Notes

About Detail Views

Menu

About Detail Views

Education

PREV NEXT

Notes

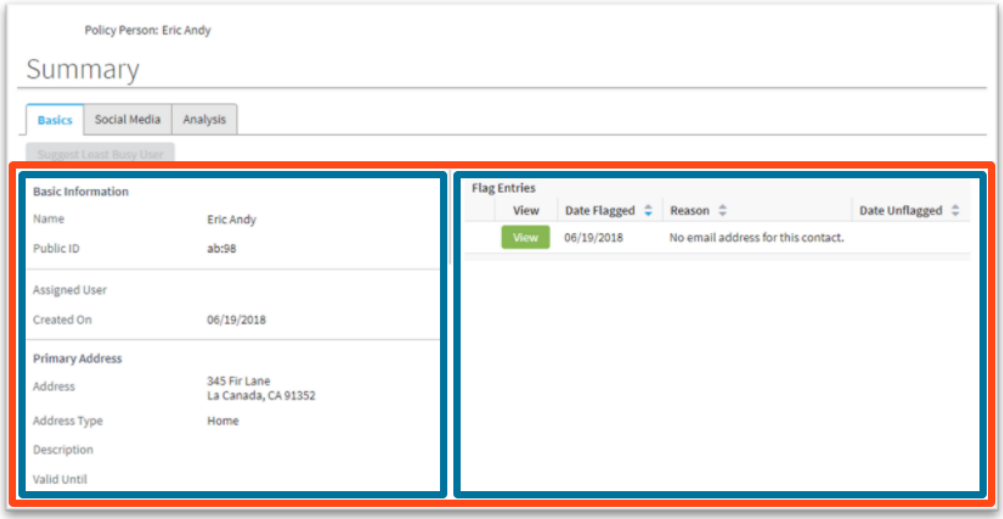
Example (4 of 4)

... but it can contain more than one to also organize the widgets horizontally in the Detail View.

To do that, we add a second Input Column.

Menu

Example (4 of 4)



- Detail View must have at least one Input Column
- Input Columns organize layout and input widgets

Education

PREV NEXT

GUIDEWIRE

Notes

Inline widget vs. PCF file

When a container such as a detail view panel is a top-level container, it is reusable. A top-level container is a PCF file. If the container is likely to be needed in multiple places, create a PCF file for the container. A Detail View Panel is PCF file that is ideal for multiple references. Other PCF files can reference the reusable container using a reference widget. In TrainingApp, the ABContactSummaryPage contains a Screen with a Panel Ref that references ABContactSummaryDV. Panel Ref widgets are discussed later in this lesson.

When a container such as detail view panel is declared as an inline child container, it is not reusable. Other containers cannot reference an inline container. A Detail View widget is defined in a Screen, Card, or a List Detail Panel. It is possible to define a variable for a Detail View widget object, but this is uncommon. A Detail View widget inherits the root object associated with its parent.

Inline widget vs. PCF file

Detail View widget	Detail View PCF file
• Differences:	• Differences:
• Not reusable	• Reusable
• Inherits parent root object	• Takes a root object
• Created as a Detail View widget in the parent container's PCF file	• Created as separate PCF file and filename ends with DV
• Similarities:	• Similarities:
• It can exercise control over all elements	• It can exercise control over all elements

If the container is likely to be needed in multiple places, create a reusable PCF file

Education

Notes

About Atomic Widgets

Menu

About Atomic Widgets

Education

Notes

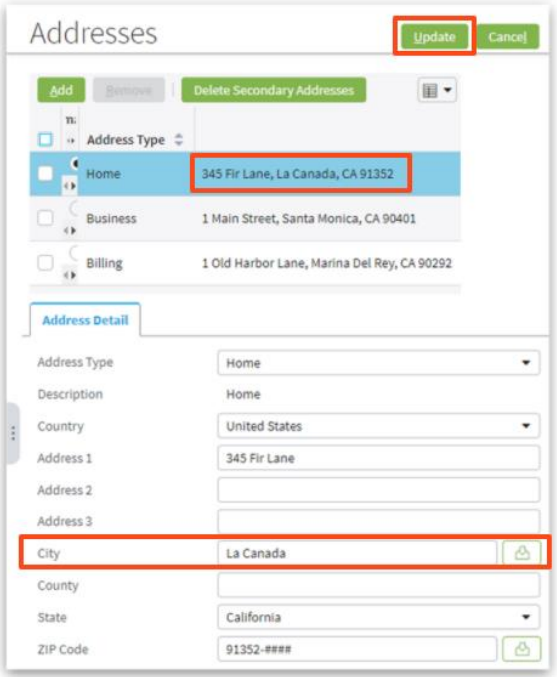
Atomic widgets

A widget is a PCF element that a Guidewire application renders into an HTML object. Buttons, menus, text boxes, and dropdown lists are all examples of PCF elements that are considered widgets. Some widgets that are not visually presented in the web browser, but are rendered as an HTML object, such as a hidden input.

Display key is a Guidewire application mechanism that allows locale-specific texts to be dynamically rendered in the browser depending on the user's internationalization language settings. Every key has one or more localized text values. When a PCF file references a display key, the application converts the key to one of the localized values depending on the user's language settings.

Atomic widgets

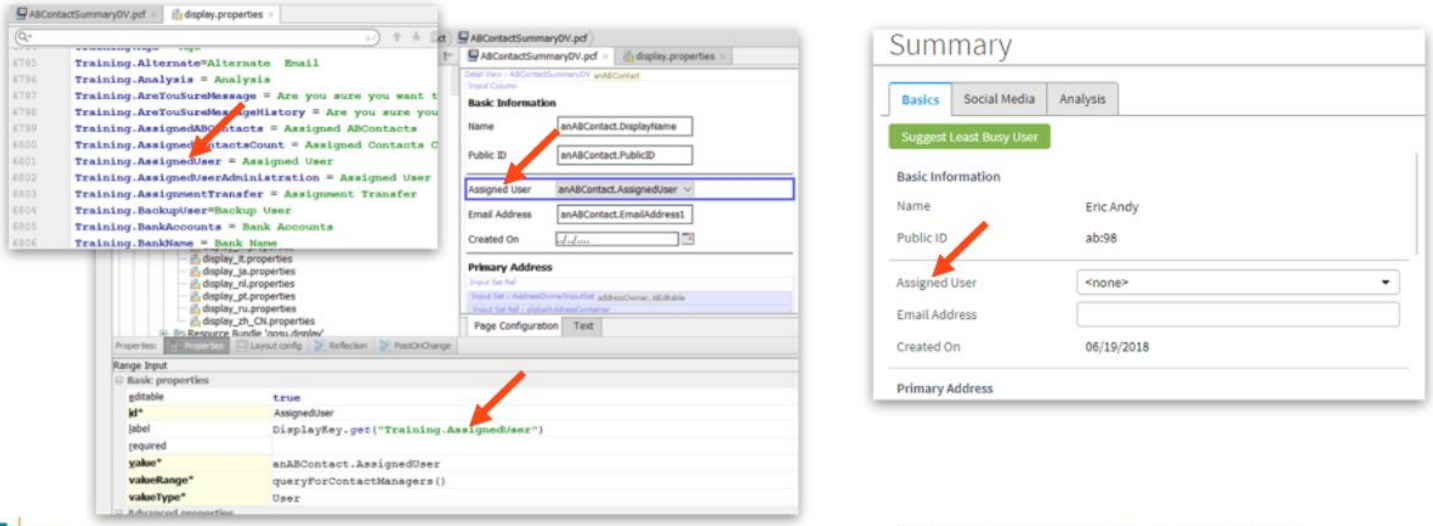
- An **atomic widget** is a graphical user interface element that is non-divisible
- An atomic widget displays individual data values and/or executes individual actions
- Input and cell widgets bind data values
  - Data values often associated with root objects, query objects, or related objects



Education

Display keys and widgets

- Widgets often include a label rendered with them in the UI
  - It's a recommended practice to use Display Keys to display static text in the UI
  - Display Keys support easy text reusability



Education

Notes

What must you specify in a widget's "value" property?

Menu

Answer

What must you specify in a widget's "value" property?

You must specify the field in the data model that the widget is bound to.

Education

PREV NEXT

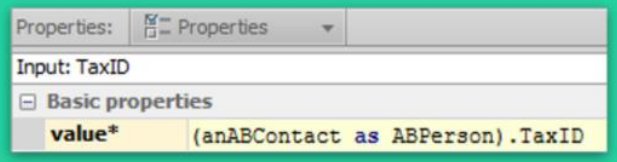
Notes

When would you see the keyword "as" in the value property?

Menu

Answer

When would you see the keyword "as" in the value property?



You would see the "as" when the container's base object is subtyped, and the field to which the widget must be bound is at one of the subtype levels.

Education

PREV NEXT

GUIDEWIRE

Notes

What is a display key?

Menu

Answer

What is a display key?

A display key is an abstract display value referenced by a widget's label property. It can have one or more language-specific values, and when the UI is rendered, the value matching the user's language is used.

Education

PREV NEXT

Notes

If you create a widget and specify only the ID, name, and label, is the widget visible? Editable? Required?

Menu

Answer

If you create a widget and specify only the ID, name, and label, is the widget visible? Editable? Required?

The widget will be visible, not editable, and not required.

Education

PREV NEXT

Notes

Configuring the Add and Remove buttons

Menu

Configuring the Add and Remove buttons

Education

PREV NEXT

Notes

Iterator Buttons widget

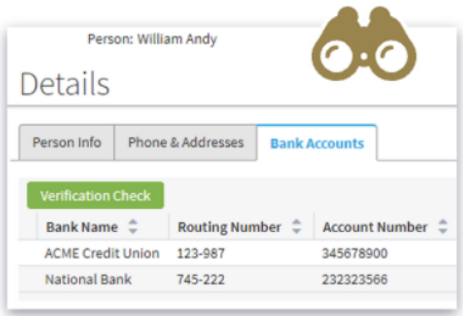
The buttons already have the appropriate visibility and availability logic:

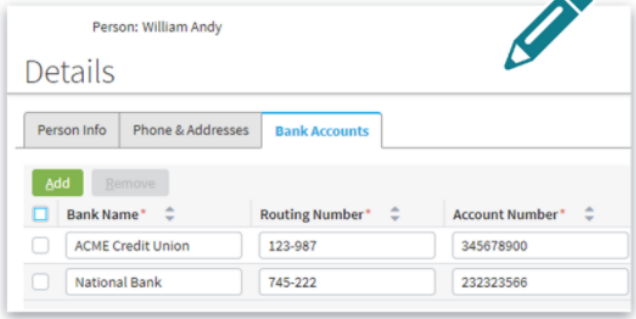
- Both buttons are visible only if the screen is edit mode
- Both buttons are available only if their actions are defined
- Remove button is available only if its actions is defined AND at least one row is selected in the List View

Menu

Iterator Buttons widget

- Iterator Buttons widget contains two buttons: Add and Remove
- It provides functionality
  - To add new rows with the Add iterator button
  - To remove existing rows with the Remove iterator button
- The buttons already have the appropriate visibility and availability logic





Education

PREV NEXT

Notes

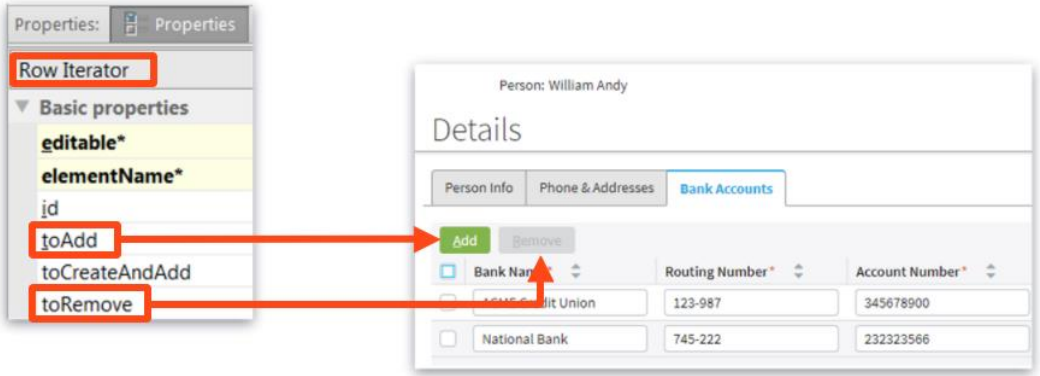
Iterator buttons and row iterators

Properties of Row Iterator govern functionality of Iterator buttons.

Menu

Iterator buttons and row iterators

- Properties of Row Iterator govern functionality of Iterator buttons (Add|Remove)
  - toAdd - action to take when Add is clicked
  - toRemove - action to take when Remove is clicked



Education

Guidewire Proprietary & Confidential – DO NOT DISTRIBUTE

Page 7

PREV NEXT

Notes

Iterator buttons and row iterators

The add button is usually configured one of the following two ways:

a) When the user clicks Add, a new empty row (object) is inserted in the List View. The user can initialize the row (object) by simply editing the cells.

b) When the user clicks Add, a popup opens. The popup creates an empty object and displays all the fields of the object. The user can initialize the object by specifying the different fields in the popup. When the user clicks OK, the initialized object will be inserted to the List View and displayed as a row.

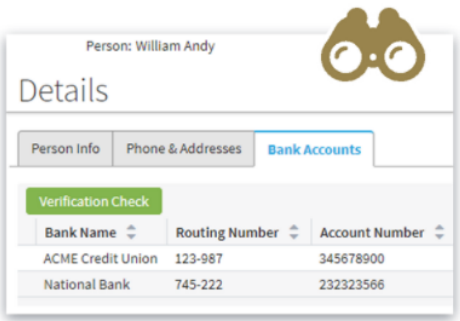
The first one is typically used when the List View displays all the fields that need to be initialized. The second approach is used when the List View displays only a few important fields of the object, but the object has more fields that need to be initialized during creation. In the example, the List View displays 3 fields of the Vendor Evaluation, but the object has other fields that may need to be initialized, so going with the second approach seems to be the better solution in this case.

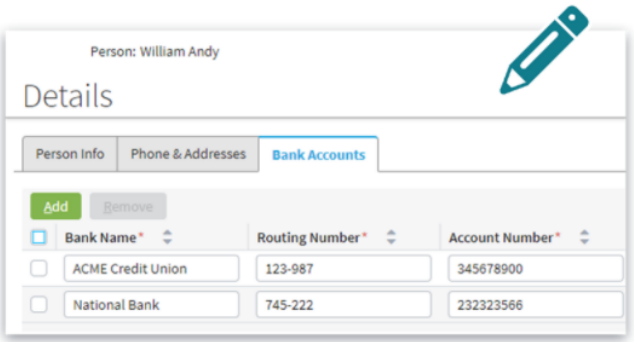
Note: To implement this second use case, you would also need to configure a creational popup. Creating objects with popups is out of scope for this course.

Iterator buttons and row iterators

Typical configuration

- The typical method for the Add button is shown here, which is used when the List View displays all the fields that need to be initialized
- The user clicks the Update button, which puts the screen in edit mode
- The user clicks the Add button, and a new empty row is inserted in the List View. The user can initialize the row by simply editing the cells.





Education

PREV NEXT

Notes

Editable hierarchy

The following summarizes the default value of the editable property for each element:

A **list view** panel's editable property is blank. The property is not required. (A list view panel with no specified editable property is editable.)

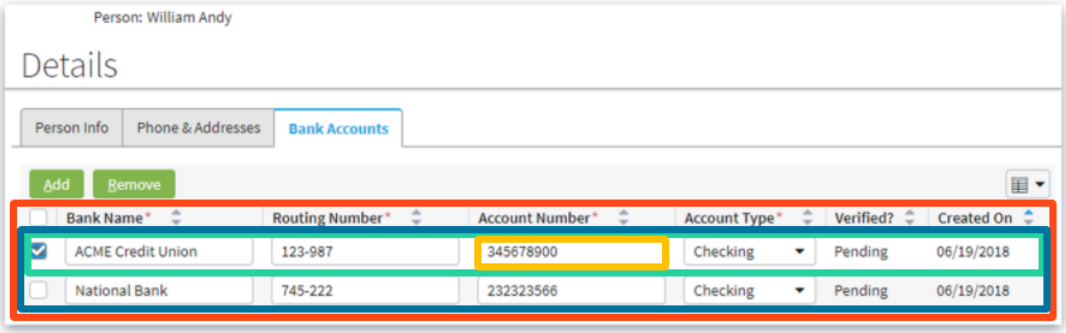
A **row iterator's** editable property is blank, but this property is required. To have editable cells, it must be set to true, or to a condition that evaluates to true.

A **row's** editable property is blank. The property is not required. (A row with no specified editable property is editable.)

A **cell's** editable property is false. To have the cell be editable, you must set the property to true, or to a condition that evaluates to true.

In practice, the two elements you must explicitly set to editable to get editable cells are the row iterator and the cells that are to be made editable.

Editable hierarchy



List View panel is editable
Row Iterator is editable
Row is editable
Cell is editable

- For a cell to be editable, only the row iterator needs to be explicitly set to editable
  - List View Panel and Row can inherit editable from parent
- List view panels can include editable cells and non-editable cells
  - Routing Number is editable
  - Created On is read-only

Education

Notes

When you drag an Iterator Buttons widget onto a toolbar, how many buttons are added? (Answer)

Menu

Answer

When you drag an Iterator Buttons widget onto a toolbar, how many buttons are added?

Two: Add and Remove

Education

Notes

The List View Panel Structure

Menu

The List View Panel Structure

Education

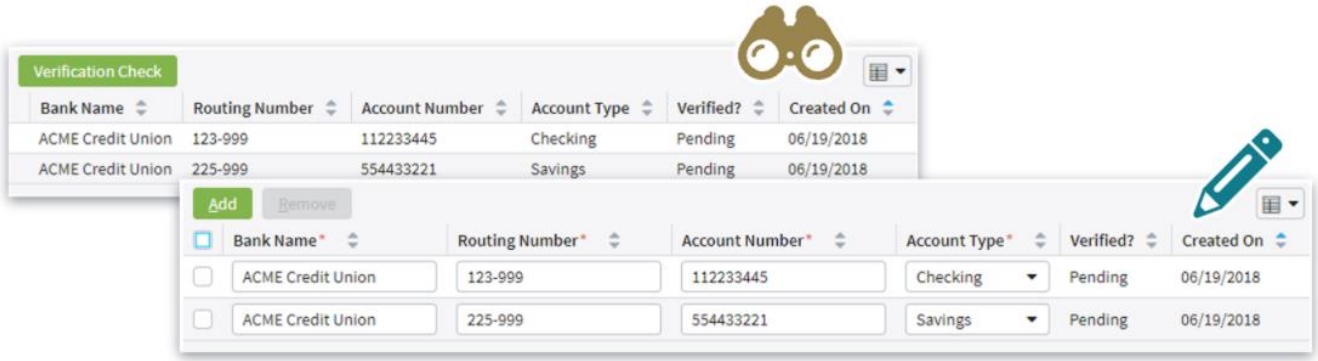
PREV NEXT

Notes

List View Panels

Just as a detail view panel is designed to allow the user to view detailed information about a single object, a list view panel is designed to allow the user to view summary information about a collection of objects.

List View Panels



- A List View Panel is a container widget that allows the user to view information about multiple objects at the same time
- Often referred to as "List View“
- In certain cases, users can also edit the data displayed in the List View

Education

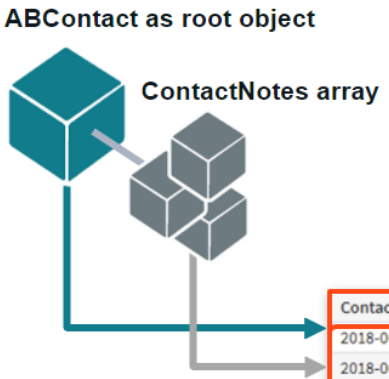
Notes

Typical List View Panel structure

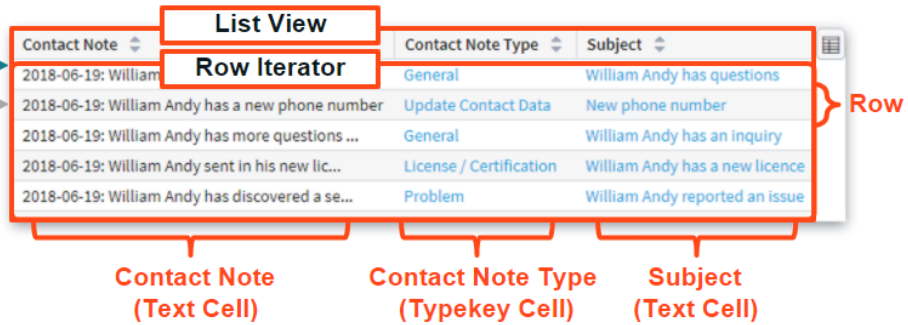
Row widgets organize the layout of the cells for an object instance when the row iterator is manipulating that instance. Row widgets do not perform any "navigation" within the collection of object instances. Adding a second row widget would not display two different objects when the page is rendered but would arrange the cells for each instance across two rows. The generated markup for a row widget is a <TR> HTML tag.

The List View's root object typically the parent of the array and not the array itself. This is because the parent entity automatically provides the addTo<ArrayName> and removeFrom<ArrayName> functions that we can use the make the List View later editable. In other cases, the root object could be the set of elements that needs to be displayed. For example, the List View displays the results of a database query. In this case there is not parent object, thus there are not addTo and removeFrom functions.

Typical List View Panel structure



- List View Panel typically takes single root object that has an array of entities to process
- Row Iterator processes the related array and produces a row for each entity instance
- Row contains one or more Cell widgets
- Cell displays an individual field of an entity



Education

Notes

Row Iterator

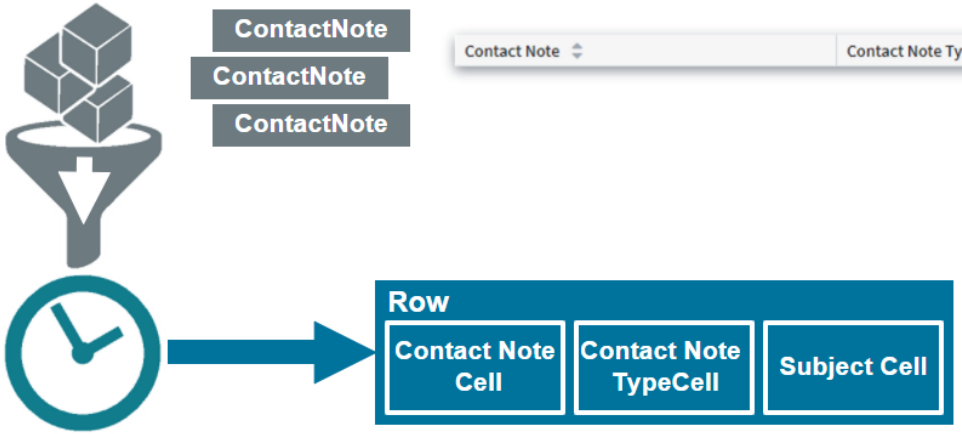
The PCF architecture has several different types of iterators, including:

- Menu item iterators, which take a set of objects and generate one menu item for each.
- Panel Iterators, which take a set of objects and generate one panel (typically, one detail view) for each.
- PolicyCenter Coverage Iterators, which take a set of coverages associated with a covered item (such as a vehicle) and generate one coverage input for each coverage.

Menu

Row Iterator

- An **Iterator** is a widget that takes a set of items and performs the same set of actions for each member
- A **Row Iterator** takes a set of objects (array or query results) and renders each object as one row of cells



Education

PREV NEXT

GUIDEWIRE

Notes

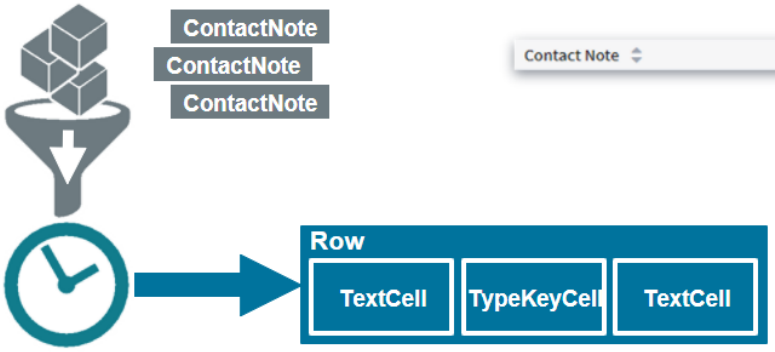
Row iterator: no objects processed

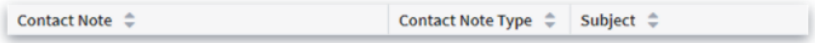
The Row Iterator always knows which object is associated with the current row. You can simply use the variable defined in the elementName property whenever you have to reference the object in the current row (e.g. configuring a cell's value/action property).

Menu

Row iterator: no objects processed

- Row Iterator has 4 required properties:
  - **value**: anABContact.HistoryEntries – the set of elements to process
  - **valueType**: HistoryEntry[] – the type of the set of elements
  - **elementName**: currentHistoryEntry – points to the object in the array associated with the row
  - **editable**: false – whether the cells can be edited or not





Education

PREV NEXT

GUIDEWIRE

Notes

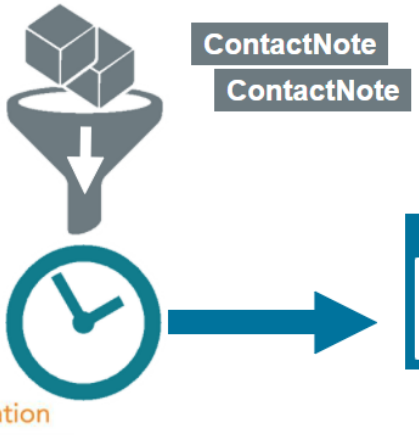
Row iterator: first object processed

When the first object is processed, the elementName points to the first object in the array.

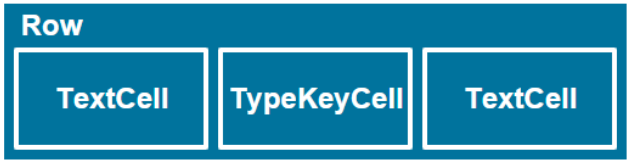
Menu

Row iterator: first object processed

- Row Iterator has 4 required properties:
  - **value**: anABContact.HistoryEntries
  - **valueType**: HistoryEntry[]
  - **elementName**: currentHistoryEntry – points to the **first** object in the array
  - **editable**: false



Contact Note	Contact Note Type	Subject
2018-06-19: William Andy has many general qu...	General	William Andy has questions



Education

Page 8

PREV NEXT

Notes

Row iterator: second object processed

When the second object is processed, the elementName points to the second object in the array.

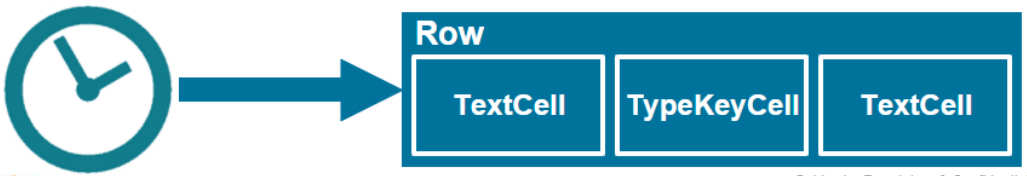
Menu

Row iterator: second object processed

- Row Iterator has 4 required properties:
  - **value**: anABContact.HistoryEntries
  - **valueType**: HistoryEntry[]
  - **elementName**: currentHistoryEntry – points to the **second** object in the array
  - **editable**: false

ContactNote

Contact Note	Contact Note Type	Subject
2018-06-19: William Andy has many general qu...	General	William Andy has questions
2018-06-19: William Andy has a new phone number	Update Contact Data	New phone number



Education

PREV NEXT

GUIDEWIRE

Notes

Row iterator: final object processed

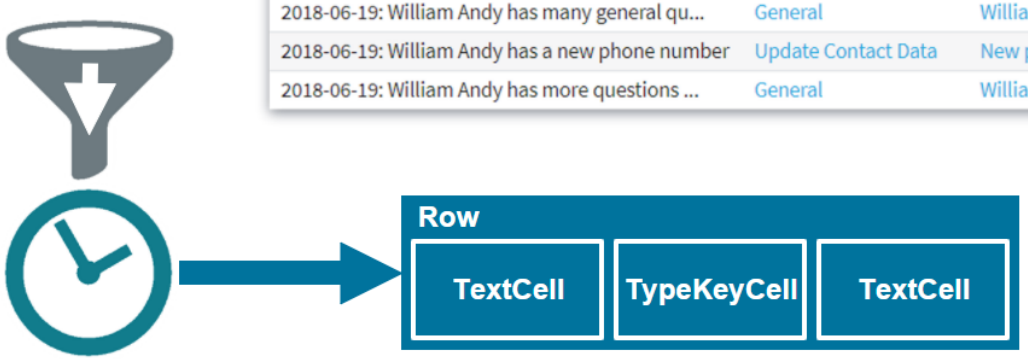
When the final object is processed, the elementName points to the third object in the array.

Menu

Row iterator: final object processed

- Row Iterator has 4 required properties:
  - **value**: anABContact.HistoryEntries
  - **valueType**: HistoryEntry[]
  - **elementName**: currentHistoryEntry – points to the **third** object in the array
  - **editable**: false

Contact Note	Contact Note Type	Subject
2018-06-19: William Andy has many general qu...	General	William Andy has questions
2018-06-19: William Andy has a new phone number	Update Contact Data	New phone number
2018-06-19: William Andy has more questions ...	General	William Andy has an inquiry



Education

PREV NEXT

GUIDEWIRE

Notes

Inline widget vs. PCF file

To enable a container widget such as List View to be reusable, it needs to be defined as a top-level container in a PCF file. This means that a top-level container itself is a PCF file. If the container is likely to be needed in multiple places, create a PCF file for the container. A List View Panel is PCF file that is ideal for multiple references. Other PCF files can reference the reusable container using a reference widget. In TrainingApp, the ABContactHistoryPage contains a Screen with a Panel Ref that references ABContactHistoryLV. PanelRef widgets are discussed later in this lesson.

When a container such as List View panel is declared as an inline child container, it is not reusable. Other containers cannot reference an inline container. A List View Panel widget is defined in a Screen, Card View Panel, or a List Detail Panel. It is possible to define a variable for an ListViewPanel widget object, but this is uncommon. A List View panel widget inherits the root object associated with its parent.

Inline widget vs. PCF file

List View PCF File

- Differences:
  - Reusable
  - Takes a root object
  - Created as separate PCF file and filename ends with LV
- Similarities:
  - It can exercise control over all elements

List View Widget

- Differences:
  - Not reusable
  - Inherits parent root object
  - Created as a List View widget in the parent container's PCF file
- Similarities:
  - It can exercise control over all elements

If the container is likely to be needed in multiple places, create a reusable PCF file

Education

Notes

In a List View:

Menu

Answer

In a List View:

A) What type of widget displays individual fields of data?

B) What type of widget organizes the individual fields of data?

A) Cell widgets

B) Row widgets

Education

Notes

A list view typically needs a toolbar, even if it is read-only. Why? (Answer)

Menu

Answer

A list view typically needs a toolbar, even if it is read-only. Why?

The toolbar is needed for the paging controls. These controls are used to view each page of rows if the number of rows is greater than what can be displayed at one time.

Education

Notes

In what way is embedding a List View Panel in a detail view panel different from embedding a list view panel in a screen? (Answer)

Menu

Answer

In what way is embedding a List View Panel in a detail view panel different from embedding a list view panel in a screen?

To embed a list view panel in a detail view panel , use a List View Input widget. To embed a list view panel in a screen (or card view panel or List Detail Panel , you use a Panel Ref widget.

Education

PREV NEXT

GUIDEWIRE

Notes

Input Sets

Menu

Input Sets

Education

PREV NEXT

Notes

Review: Input Sets

Note: It is important that Input Sets are not referenced by secondary views, don't have a toolbar directly associated with them, and are embedded into a Detail View Panel.

Menu

Review: Input Sets

- Set of atomic widgets grouped together
  - Could be reused together
  - Common visibility / editability condition could be applied across multiple widgets
  - The input set could be targeted with a partial page update to improve UI performance
- Input Sets fit some attributes of the primary containers
  - They can contain atomic widgets and organize them into logical groups
- BUT:
  - Input Sets must be embedded into Detail View Panels
  - Cannot be referenced by secondary views
  - Cannot have a toolbar directly associated with them

Education

Notes

Widgets for Input Sets

Unlike detail view panels, input sets cannot contain columns.

Menu

Widgets for Input Sets

- Input sets can contain
  - Input widgets, input dividers, labels, list view panels and other input sets
- Input sets cannot contain
  - Input columns

Education

Page 6

PREV NEXT

Notes

What are the primary use cases for input sets? (Answer)

Menu

Answer

What are the primary use cases for input sets?

Possible answers: (1) Reuse of widgets across detail view panels, (2) or logic shared for a group of widgets such as editability and/or visibility (3) or targeted page update.

Education

Notes

Modes

Menu

Modes

Education

PREV NEXT

Notes

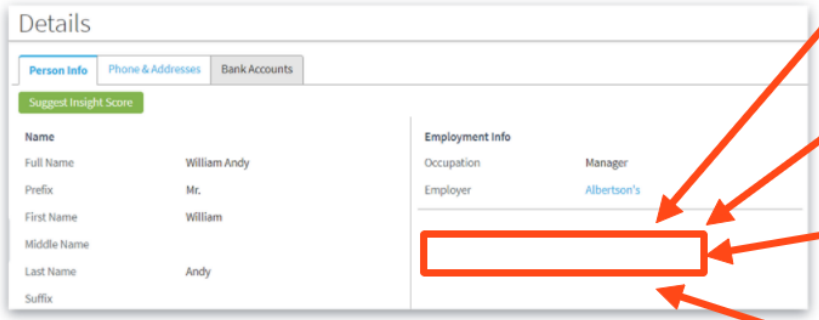
Widgets that need to vary

A common scenario in implementations is changing the appearance of a widget in the UI based on business requirements.

Menu

Widgets that need to vary

- In some cases, widgets need to vary significantly based on business scenario
- How could the requirement be implemented?



if contact is a person

License Info	
Driver's License	1213145
State	Alaska

if contact is an adjudicator

License Info	
Adjudicator License	21-11-14141313
Domain	Federal

if contact is an attorney

License Info	
Law License	FH1-HG2-113924
Specialty	General Liability

if contact is a doctor

Medical Specialty	
Category	Surgery
Specialty	Hospitalist

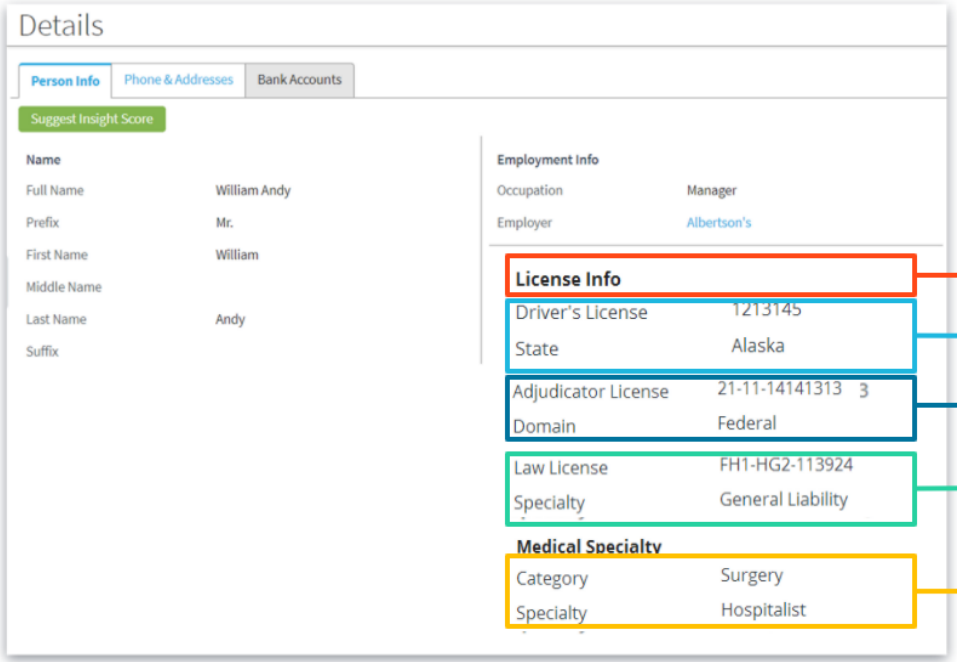
Education

Notes

Addressing all scenarios within one PCF

For user interfaces that need different structures, you could include all the possible structures in a single PCF file. Including all the possible variants in a single PCF file adds to the complexity of the PCF and eliminates the possibility of reusable containers. In addition, much of the conditional logic becomes overly repetitive. Developers of applications prefer working with smaller components: they are easy to create, debug, modify, and maintain.

Addressing all scenarios within one PCF



- Set visible attribute of widgets or containers
- All fields in one container makes container difficult to maintain

if contact is attorney, adjudicator, or person

if contact is person

if contact is adjudicator

if contact is attorney

if contact is doctor

Education

PREV NEXT

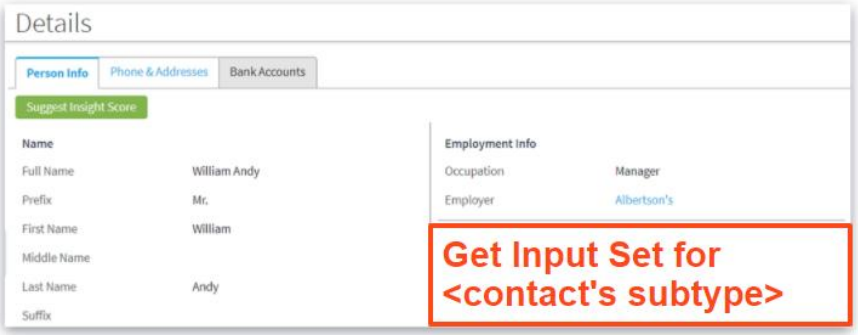
Notes

Addressing scenarios with multiple PCFs

When a single PCF cannot easily accommodate a variety of business cases, it is often easier to create a set of PCFs, one for each use case. Each version needs a value to identify the use case it is designed for. When the PCF is called, it is called by both its name and its "use case". Modes exist in the PCF architecture to implement this "use case" versioning.

Menu

Addressing scenarios with multiple PCFs



- Sometimes, best approach is to use a series of separate PCFs, each for a specific scenario
  - The application can reference the PCF appropriate to a given scenario
- Application can dynamically select and include the appropriate PCF file from the set

Input Set for ABPerson

License Info	
Driver's License	1213145
State	Alaska

Input Set for ABAdjudicator

License Info	
Adjudicator License	21-11-14141313
Domain	Federal

Input Set for ABAttorney

License Info	
Law License	FH1-HG2-113924
Specialty	General Liability

Input Set for ABDoctor

Medical Specialty	
Category	Surgery
Specialty	Hospitalist

Education

PREV NEXT

GUIDEWIRE

Notes

Modes

There is no requirement for modal PCFs to come in sets. It is possible to create a single PCF with a mode that had no counterparts. This sort of PCF would behave in exactly the same way as a PCF that didn't use modes, which would mean there was no benefit to having made the PCF modal. Therefore, in practice, modal PCFs always come in sets of PCFs.

A modal PCF can have multiple modes. In the slide example, the first SubtypeInfoInputSet is used for contacts of subtype ABPerson or ABPolicyPerson.

Modes

- **Mode** is a property used for PCFs appropriate for a given business scenario
  - Mode identifies which scenario(s) the PCF is for
  - Modal PCFs always come in sets

- Every PCF must have:
  - Same ID
  - Same number, type, and order of required variables
  - One or more unique modes (runtime error occurs if referenced mode belongs to multiple PCFs in same set)

SubtypeInfoInputSet
mode = ABPerson, ABPolicyPerson

License Info	
Driver's License	2WED458
State	Washington

SubtypeInfoInputSet
mode = ABAttorney

License Info	
Law License	JX3-AK7-309921
Specialty	Workers' Compensation

SubtypeInfoInputSet
mode = ABAdjudicator

License Info	
Adjudicator License	23-44-231223
Domain	Municipal

SubtypeInfoInputSet
mode = ABDoctor

Medical Specialty	
Category	General Care
Specialty	Internal Medicine

Education

Notes

Default mode

In the slide example, the default PCF is blank. If a SubtypeInfoInputSet is referenced but the mode it references cannot be found (such as, for example, a reference to SubtypeInfoInputSet with the mode ABPropertyInspector), then the default PCF is used.

In some situations, the "default" mode is assigned to a PCF that has other non-default modes. For example, since all person contacts are either of type ABPerson or one of its children, it would be possible to have assigned ABPerson, ABPolicyPerson, and default to the first SubtypeInfoInputSet.

Default mode

- Modal PCF sets include one PCF with mode of "default", which is used when something references a mode that doesn't otherwise exist
- "default" must be lower case
  - Runtime error occurs if unknown mode is called and no "default" widget exists

SubtypeInfoInputSet
mode = default

SubtypeInfoInputSet
mode = ABPerson, ABPolicyPerson

License Info	
Driver's License	2WED458
State	Washington

SubtypeInfoInputSet
mode = ABAttorney

License Info	
Law License	JX3-AK7-309921
Specialty	Workers' Compensation

SubtypeInfoInputSet
mode = ABAdjudicator

License Info	
Adjudicator License	23-44-231223
Domain	Municipal

SubtypeInfoInputSet
mode = ABDoctor

Medical Specialty	
Category	General Care
Specialty	Internal Medicine

Education

Notes

Common uses of modal PCFs

In PolicyCenter, modes are also used to accommodate variations in policy transaction type (submission, change, renewal, cancelation, and so on).

Menu

Common uses of modal PCFs

- Modes can be used with:
  - Detail view panels
  - Card view panels
  - Input sets
  - List view panels
  - Info bars
  - Screens
  - ...and more (refer to documentation)
- Modes are often used to accommodate variations in:
  - Contact type (individuals, companies, and so on)
  - Line of business (auto, workers' comp, property, and so on)

Education

PREV NEXT

GUIDEWIRE

Notes

Modes and subtypes

In the slide example, the first modal PCF is the default mode, and will be used for any contact whose type is not ABAttorney, ABPerson, or ABPolicyPerson.

The second modal PCF has one mode, ABAttorney, and will be used for any contact whose type is ABAttorney.

The third modal PCF has two modes, ABPerson and ABPolicyPerson, and will be used for any ABPerson or ABPolicyPerson contact.

For the diagram shown, the default mode will be used for contacts of type "ABPersonVendor" and "ABDoctor".

In the base application, there is actually an ABPersonVendor subtype in between the ABPerson level and the ABAttorney or ABDoctor level.

Modes and subtypes

- Mode property recognizes explicitly specified subtypes
  - Not aware of a subtype's hierarchy
  - Unspecified subtypes will not render
- In this example, the default mode will be selected for ABPersonVendor and ABDoctor

SubtypeInfoInputSet
mode = default

SubtypeInfoInputSet
mode = ABAttorney

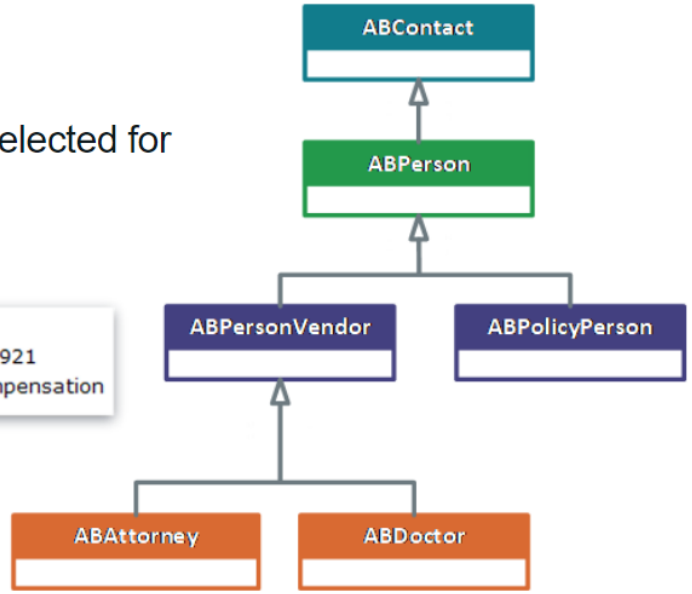
License Info

Law License	JX3-AK7-309921
Specialty	Workers' Compensation

SubtypeInfoInputSet
mode = ABPerson | ABPolicyPerson

License Info

Driver's License	2WED458
State	Washington



Education

PREV NEXT

Notes

Name two business situations in which it might be useful to use modal PCFs.

Menu

Answer

Name two business situations in which it might be useful to use modal PCFs.

Possible answers: Different types of contacts (individual vs. business); different lines of business (auto vs. property).

Education

PREV NEXT

Notes

For a given set of modal PCFs, what two things must all of the PCFs have in common? (Answer)

Menu

Answer

For a given set of modal PCFs, what two things must all of the PCFs have in common?

All of the files must have the same required variables (identical in number, order, and type), as well as the same ID.

Education

Notes

Configuring navigation widgets

Menu

Configuring navigation widgets

Education

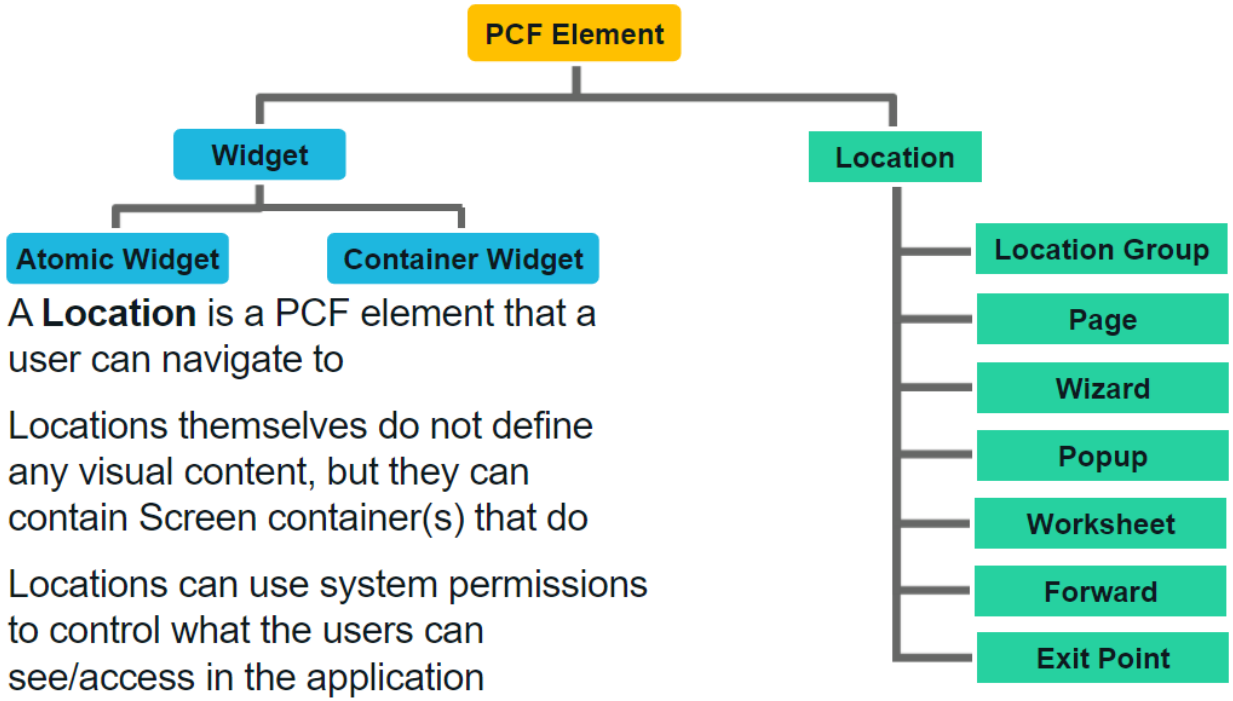
Notes

Locations

Note: Both Widget and Location are conceptual representations in this diagram. There are no <Widget /> or <Location /> elements. Similarly, both Atomic Widget and Container Widget are conceptual representations. There are no <Atomic Widget /> or <Container Widget /> elements.

Menu

Locations



- A **Location** is a PCF element that a user can navigate to
- Locations themselves do not define any visual content, but they can contain Screen container(s) that do
- Locations can use system permissions to control what the users can see/access in the application

Education

PREV NEXT

GUIDEWIRE

Notes

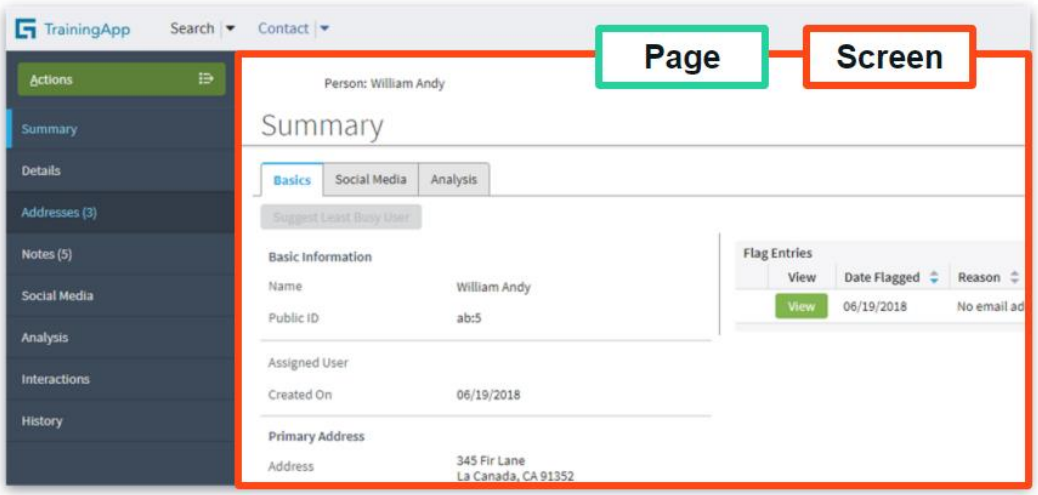
Pages

For more information, see the topic, "Introduction to Page Configuration" in the product documentation.

Menu

Pages

- A **Page** contains a single screen
  - Pages are used exclusively within Location Groups



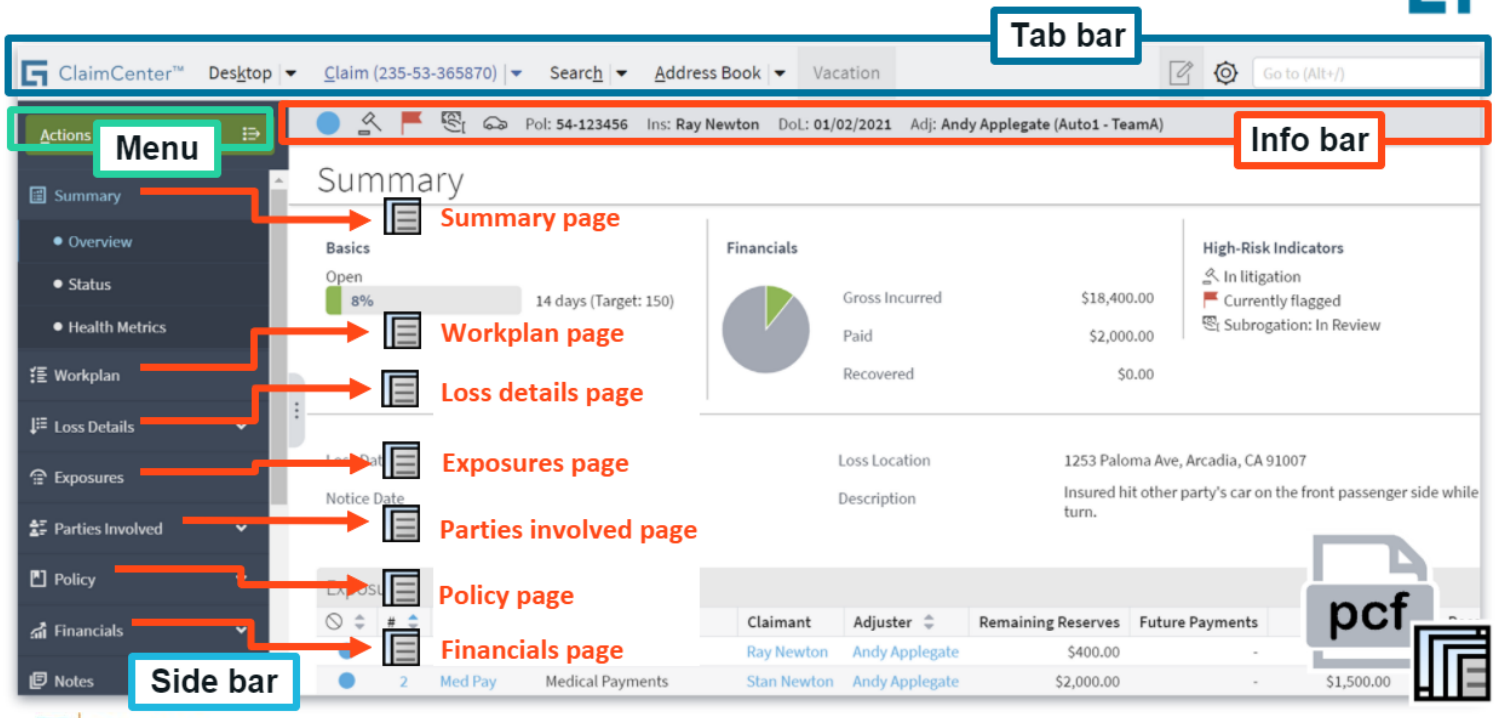
Education

Notes

Location groups

A Location Group can be thought of as a "page group" as it is fundamentally a collection of pages, each with its own screen. Location groups are used to gather together a set of screens that display data about a single primary object (such as a contact, a policy, an account, or a claim) or serve a single major application function (such as searching for data). In the example on the screen, the location group displays data about a claim. Users navigate from one page in a Location Group to the next by clicking the links in the side bar.
All the pages in a Location Group share a common info bar, actions menu, and side bar.
The tab bar appears at the top of the screen and allows the user to navigate to different parts of the application. For example, to view another claim from the Claim dropdown menu, to go to an appropriate search screen or the address book, and so on.
The info bar is the area directly below the tab bar that usually contains icons providing high-level information about the data in the screen area. In the example above, the info bar contains a "car" icon indicating this is a personal auto claim. It shows the name of the insured "Ins: Ray Newton", and the date of loss "DoI: 01/02/2021". Note that

Location groups



Education

PREV NEXT

Notes

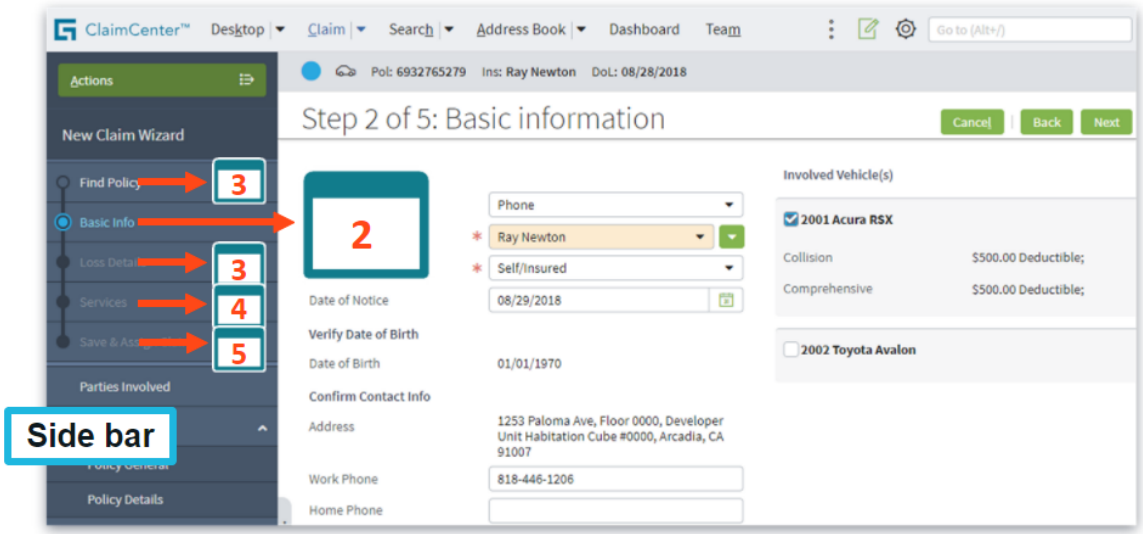
Wizards

A Wizard has multiple screens, but only one screen is displayed at a time. The screens in a Wizard have an order (though it may be possible for a user to traverse the screens out of order).

The Wizard shown above does not exist in an application. Wizards are implemented differently in each business-oriented Guidewire application. Multiple examples of wizards appear on the following slides, one for each of the primary applications.

Wizards

A **Wizard** is an ordered collection of screens used to execute a complex business process



- Single info bar, actions menu, and side bar
- Includes toolbar with Back and Next buttons

Education

PREV NEXT

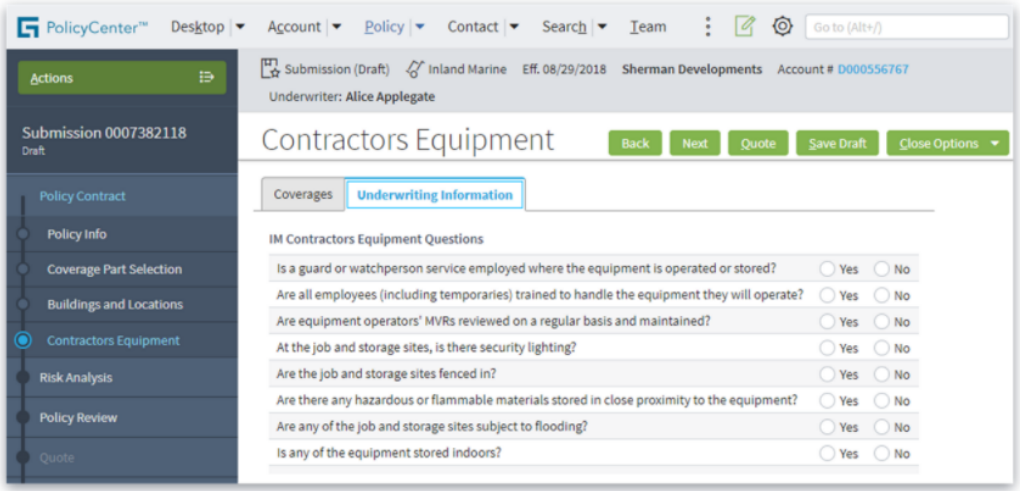
Notes

Wizard example: PolicyCenter

The Wizard in the slide example is the Submission Wizard, which is used to create and modify submissions. A submission ends when it is withdrawn or bound. The outcome of successfully binding a submission is a policy.

Wizard example: PolicyCenter

PolicyCenter wizards are used to create, modify, renew, or cancel policies



Education

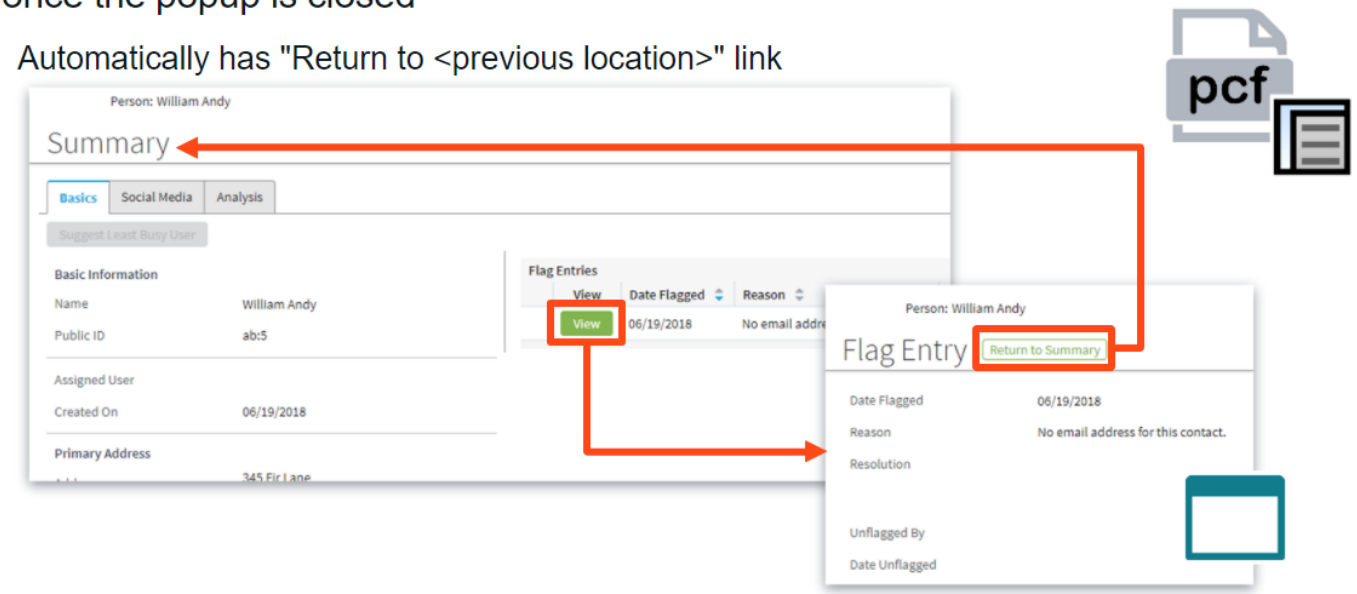
Notes

Popups

Popup screens are designed to mimic the functionality of a true popup (which is an entirely separate window). True popups are difficult to architect because there is no easy way to avoid synchronization errors in which the system tries to update the same object in two separate windows. Popup Locations as architected in Guidewire offer virtually the same functionality while avoiding the synchronization and usability issues.

Popups

- A **Popup** contains a single screen and returns the user to the previous Location once the popup is closed
  - Automatically has "Return to <previous location>" link



Education

PREV NEXT

Notes

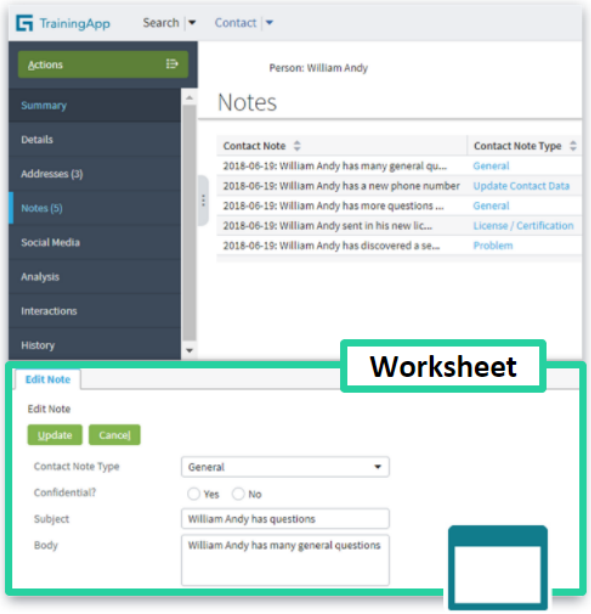
Worksheets

The worksheet is an area of the user interface that runs across the bottom. It is visible only when a screen is displayed within it. It is the one area of the user interface that is not always visible.

If multiple worksheets are rendered at one time, the tabs across the top of the worksheets can be used to navigate between worksheets.

Worksheets

- A **Worksheet** contains a single screen rendered in the workspace frame



Education

Notes

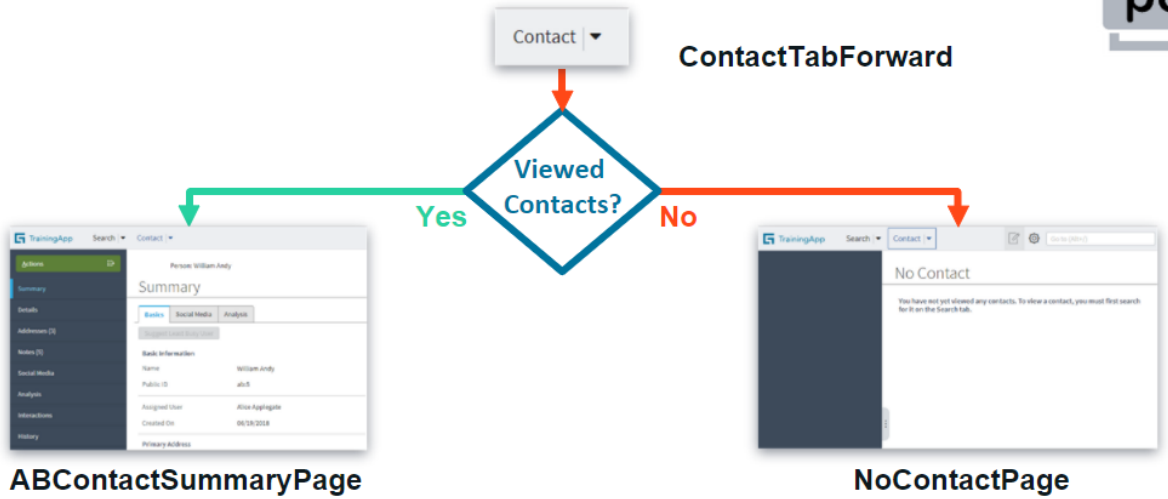
Forwards

You can use forwards to:
- Modify data before navigating
- Determine the destination Location based on the data context or the user's permissions

Menu

Forwards

- A **Forward** contains logic to execute before navigating to another Location
- Often involves deciding which Location to navigate to



Education

PREV NEXT

Notes

Exit points

Exit points are typically used to let users easily access other applications, such as a reporting application.

Menu

Exit points

- An **exit point** points to a URL outside of the Guidewire application
- Often used to access other applications or websites
- Does not contain (directly or indirectly) a Screen widget

Education

Notes

Review of Locations navigation

A Page contains a single screen in the screen area.

A Popup contains a single screen and is designed to return the user to the previous Location once the work on the popup screen is complete.

A Worksheet contains a single screen and a tab (on the bottom) in the workspace frame.

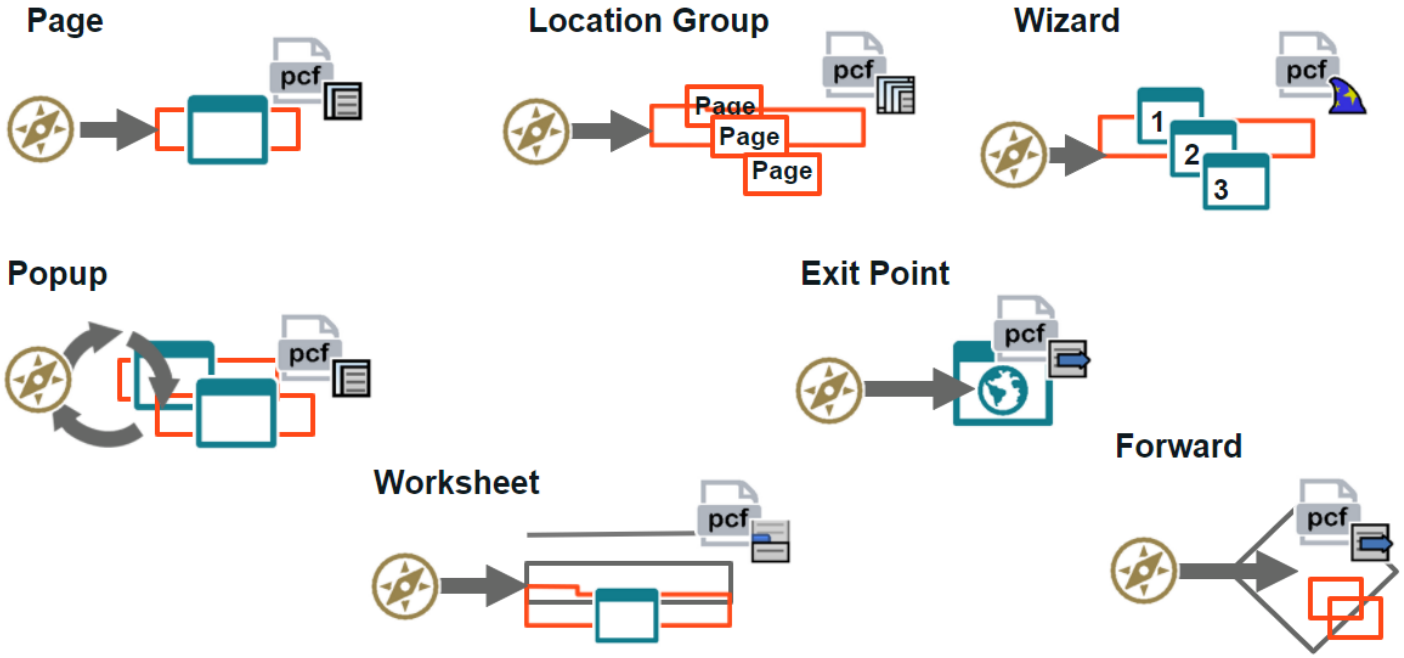
A Location Group organizes a set of menu links (and their associated Locations), a set of menu actions, and an info bar into a group.

A Wizard contains multiple screens in a specific order and a toolbar to work through the wizard.

A Forward contains logic to execute before navigating to another Location.

An Exit Point points to a URL outside of the Guidewire application.

Review of Locations navigation



Education

Notes

Locations summary

This table provides a summary of Location types, their typical navigation method, what they initially display, and what component of the UI the initial display is located in.
Note: In ClaimCenter and PolicyCenter, navigation to a wizard typically uses go(). In BillingCenter, navigation to a wizard typically uses push().

Locations summary

	Typical Navigation Method	Initially Displays	In
Forward	**go()**	Nothing	--
Location Group	**go()**	First child page	Screen area
Page	**go()**	Screen	Screen area
Wizard	**go()** **push()**	First screen	Screen area
Worksheet	**goInWorkspace()**	Screen	Workspace frame
Popup	**push()**	Screen	Originating frame
Exit Point	**push()**	External page or web site	New window (or entire existing window)

Education

Notes

Of the seven types of Locations listed... Which two contain multiple screens and have their own info bar, actions menu, and side bar?

Menu

Answer

**Of the seven types of Locations listed...
Which two contain multiple screens and have their own info bar, actions menu, and side bar?**

- Locations
- Page
- Location Group
- Wizard
- Popup
- Worksheet
- Forward
- Exit Point

The correct response is:

Location Group and Wizard

Education

PREV NEXT

Notes

Of the seven types of Locations listed... Which one renders a screen somewhere other than the screen area?

Menu

Answer

Of the seven types of Locations listed... Which one renders a screen somewhere other than the screen area?

- Locations
- Page
- Location Group
- Wizard
- Popup
- Worksheet
- Forward
- Exit Point

The correct response is:

Worksheet and Exit Point

Education

Notes

Of the seven types of Locations listed... Which one typically navigates to one of several Locations based on business logic?

Menu

Answer

**Of the seven types of Locations listed...
Which one typically navigates to one of several Locations based on business logic?**

- Locations
- Page
- Location Group
- Wizard
- Popup
- Worksheet
- Forward
- Exit Point

The correct response is:

Forward

Education

Notes

Of the seven types of Locations listed... For each Location, what method would you usually use to navigate to it?

Menu

Answer

Of the seven types of Locations listed...
For each Location, what method would you usually use to navigate to it?

- Locations
- Page
- Location Group
- Wizard
- Popup
- Worksheet
- Forward
- Exit Point

The correct responses are:

Use *go()* to navigate to Pages, Location Groups, Wizards, and Forwards.

Use *push()* to navigate to Popups and Exit Points.

Use *goInWorkspace()* to navigate to Worksheets.

Education

Notes

What are the three primary use cases for popups? (Answer)

Menu

Answer

What are the three primary use cases for popups?

Viewing and editing existing objects, creating new objects, and executing popped searches.

Education

PREV NEXT

Notes

What happens if a location's canEdit property evaluates to false? (Answer)

Menu

Answer

What happens if a location's canEdit property evaluates to false?

The screen and its contents are non-editable.

Education

PREV NEXT

Notes

What happens if a location's canVisit property evaluates to false? (Answer)

Menu

Answer

What happens if a location's canVisit property evaluates to false?

The widget navigating to the location is either not visible or not clickable.

PREV NEXT

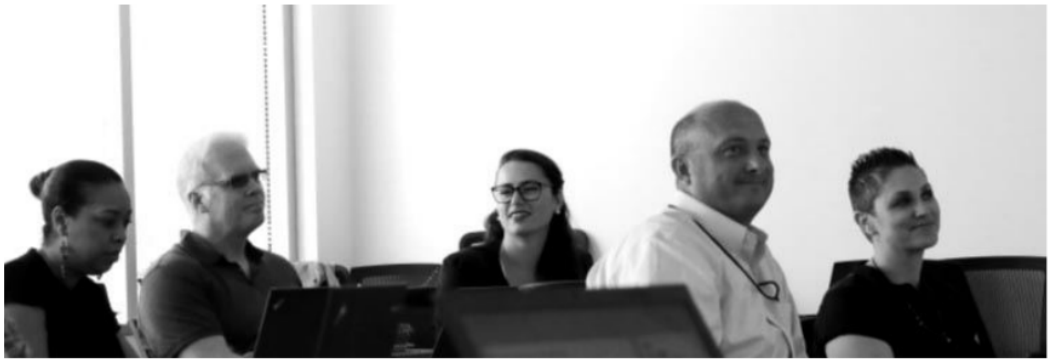
Notes

Introduction to Gosu

Menu

Education

Introduction to Gosu



NEXT ›

Notes

Using Gosu to configure the Application Tier

Menu

Using Gosu to configure the Application Tier

Education

PREV NEXT

Notes

Guidewire Gosu

Gosu is Guidewire's open-source, publicly available programming language. Gosu has elements of both procedural and object-oriented programming languages and is similar to JavaScript and Java.

Guidewire developed Gosu for several reasons.

First, there was a desire to have a single syntax that could be used to work with all of the elements relevant to Guidewire products (such as entities, display keys, classes, class enhancements, Java classes, permissions, and script parameters) even though these items have fundamentally distinct internal implementations. There was no existing language that provided this ability.

Second, there was a desire to have code auto-complete features in Studio. This is possible only with a statically typed language. Most scripting languages, such as JavaScript, Perl, Python, and Ruby, are dynamically typed.

In very early releases of InsuranceSuite, Gosu was known as GScript.

Guidewire Gosu

- Guidewire's programming language
  - Similar to Java and compatible with Java
  - Has elements of both procedural and object-oriented programming languages
- Executes fundamental application behavior
- Manages complex business processes
- Executes hierarchical Gosu Rules
- Specifies dynamic user interface behavior

Notes

Gosu in Guidewire applications (1 of 3)

A rule is a single decision in the following form of testing a condition and then taking an action: if {some conditions} then {take some action}. A rule set combines many individual rules into a useful set to consider as a group.

Gosu enhancements provide additional methods (functionality) on a Guidewire entity. For example, you can create an enhancement to the ABContact entity. Within enhancement, you add methods that support new functionality. Guidewire Studio will show the new method for an entity of the type ABContact.

Note: Enhancements are not limited to entities. Technically any Gosu or Java class can be enhanced.

Gosu in Guidewire applications (1 of 3)

Gosu Rules

- Apply specific programming logic for testing a condition and performing an action
- Examples:
  - Event Fired Rules
  - Pre-update Rules
  - Validation Rules

Entity Enhancements

- Extend entity functionality with programming logic as entity methods
- Examples:
  - ABCompany.maskTaxId() returns a value with a mask for the first 5 characters

Education

Notes

Gosu in Guidewire applications (2 of 3)

Shown here is a comparison between the use of Entity Names and Gosu Classes.

Menu

Gosu in Guidewire applications (2 of 3)

Entity Names

- Programming logic that defines how to display a name for an entity instance
- Examples:
  - Drop-down lists of contacts
  - DisplayName property of entity instance

Gosu Classes

- Encapsulate data and code for a specific purpose or function
- Examples:
  - Utility helper classes for logging or string functions
  - Plugins for integrations with other applications

Education

PREV NEXT

Notes

Gosu in Guidewire applications (3 of 3)

A workflow executes processes whose execution is time-based or is triggered by events in external applications or is both triggered and time-based.

Workflows are available in all three primary Guidewire applications. PolicyCenter uses workflows to manage policy transactions (such as renewals). BillingCenter uses workflows to manage account delinquency and agency bill cycles. Although the feature is available in ClaimCenter, the base configuration of ClaimCenter does not use workflows.

Gosu in Guidewire applications (3 of 3)

Workflows

- Run custom business processes asynchronously, optionally with multiple states that transition over time
- Examples:
  - Renewal and cancelation workflows

Notes

Gosu Scratchpad

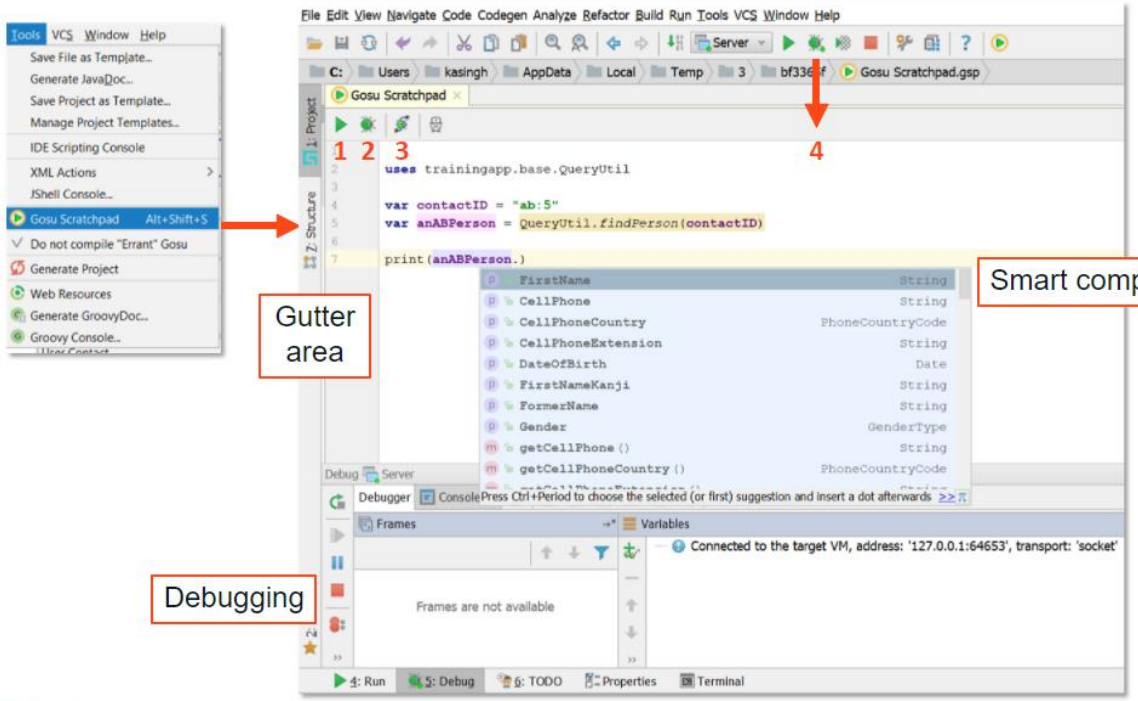
Gosu Scratchpad can be used to write, execute and debug Gosu code snippets.
To open up the Gosu scratchpad, select Tools and then Gosu Scratchpad
In the example above, we run a simple Gosu query to load data from the application database.
We can also execute transactions to commit test data.

The Gosu scratch pad supports common editor features

- Gutter Area
- Smart Completion
- Validation bar and
- Debugging

The Run in Debug Server process button (3) uses the Platform compiler while the Run (1) and Debug (2) buttons use the community (open source) compiler. There are a few minor differences between the two versions of Gosu, but those are out of scope for this course.
It is recommended to use the Run in Debug Process button (3) and use Platform compiler since all the applications are written using the Platform compiler.
The server must be running in debug mode (4) before you can run Gosu code using the debug server process button (3).

Gosu Scratchpad



1 Run

2 Debug

3 Run in Debug Server process

4 Run (Server) in debug mode

Education

Notes

Arrays

Configuration of Guidewire applications often focuses on working with data model entity arrays. An array defines a set of additional entities of the same type to associate with the main entity.

A related data model array is the associative array. An associative array provides a mapping between a set of keys and the values that the keys represent. A common example of this type of mapping is a telephone book, in which a name maps to a telephone number. Another common example is a dictionary, which maps terms to their definitions.

An array is a collection of data values, with each element of the array associated with a number or index.

In typical Gosu code, simply use angle brackets after the type name, such as String[] to represent an array of String objects. Use a zero-based index number to access an array member. If you create an array, you must explicitly define the size of the array or implicitly define the size by simultaneously defining the array elements. To access the elements of an array, you use an index expression.

Arrays

Data model entity array

- `<array/>` on parent entity to array entity
- `<foreignkey/>` on array entity to parent entity
- **Example**
  - ABContact defines FlagEntries array

Derived API entity array

- Entity enhancement or Guidewire API
- **Example**
  - ABContact AllAddresses property returns an Address array

Gosu datatype and object array

- A set of values are of the same type in a single collection
- **Examples**
  - `var a = new int[3]`
  - `var b = new int[] {1,2,3}`
  - `var c : int[] = {1,2,3}`

Education

PREV NEXT

Notes

Array methods that require logic

A variety of array methods are available in Gosu, including methods that get information about the array or that get members of the array that match a given condition.
For methods that require an argument, the argument must be an expression of a condition, such as returning *true* if any row in the array matches the condition, returning the first row matching the condition, or returning all rows that match the condition. Arguments must be in the form of a Gosu block expression.

Array methods that require logic

- Array methods include
  - Getting information about the array
  - Getting members of the array that match a given condition
- Some methods require an argument
- Argument must be an expression of a condition, such as:
  - Returns true if any row in the array matches the condition
  - Returns the first row that matches the condition
  - Returns all rows that match the condition
- Argument must be a Gosu block expression

Notes

Common array methods (1 of 2)

Common array methods include *hasMatch(condition)*, which determines if any element in an array matches the condition provided, and *countWhere(condition)*, which returns a count of elements in an array that match the condition.

Menu

Common array methods (1 of 2)

- `hasMatch(condition)`
  - Determine if any element in an array matches a given condition
  - Syntax: `array.hasMatch(\ name -> conditionToMatch)`
- `countWhere(condition)`
  - Returns count of elements that match a given condition in an array
  - Syntax: `array.countWhere(\ name -> conditionToMatch)`

PREV NEXT

Notes

Common array methods (2 of 2)

Additional array methods include *firstWhere(condition)*, which retrieves the first element of an array matching the condition, and *where(condition)*, which creates a target array consisting of the members of the source array that match the condition. For a more detailed description and information on other methods, see the product documentation.

Menu

Common array methods (2 of 2)

- `firstWhere(condition)`
  - Retrieve the first element in an array that matches a given condition
  - Syntax: `array.firstWhere(\ name -> conditionToMatch)`
- `where(condition)`
  - Retrieves a target array that consists of all the members of a source array that match a given condition
  - Syntax: `array.where(\ name -> conditionToMatch)`

PREV NEXT

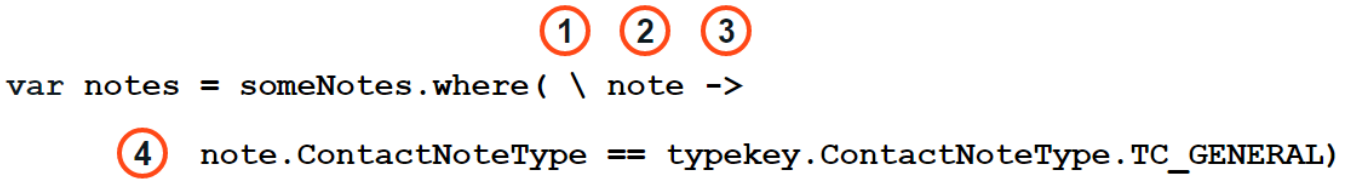
Notes

Blocks argument expression

Blocks are used in many situations where a method requires an expression as an input parameter, such as in arrays, queries, and transaction bundles for database transactions.
A block is an expression of logic passed to a method as an argument. Some array methods require conditions using a block.
The block consists of four parts:

1. \ , which identifies that the following is a block
2. An element name representing each element of the array
3. ->, which identifies the start of the condition
4. A condition, which relates to the element

Blocks argument expression



- A **block** is an expression of logic passed to a method as an argument
- Some array methods require conditions using a block
- The block consists of four parts:
  1. \ , which identifies that the following is a block
  2. An element name representing each element of the array
  3. ->, which identifies the start of the condition
  4. A condition, which relates to the element

Education

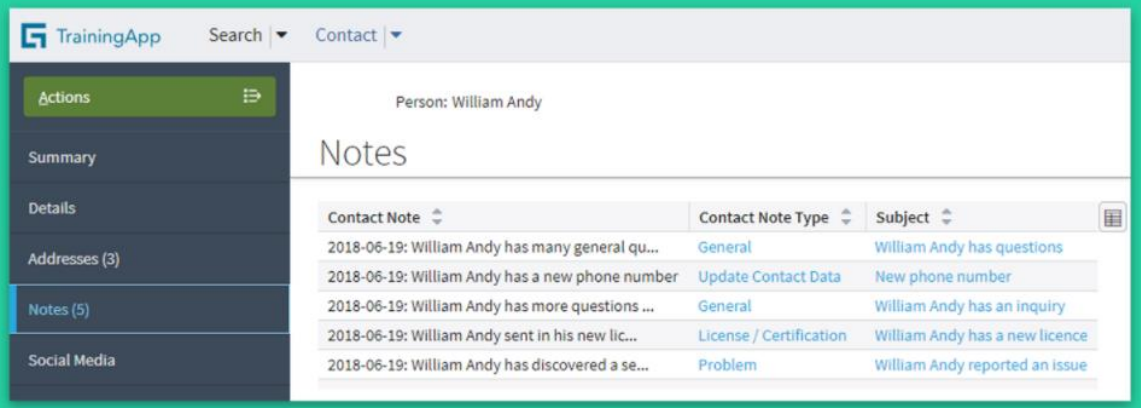
Notes

In the screenshot shown here, what is the array method to return the number of objects in the ContactNotes array? What is the value?

Menu

Answer

In the screenshot shown here, what is the array method to return the number of objects in the ContactNotes array? What is the value?




```
var count = array.length or array.Count.
```

The total number of elements in the ContactNotes array is 5.

Education

PREV NEXT

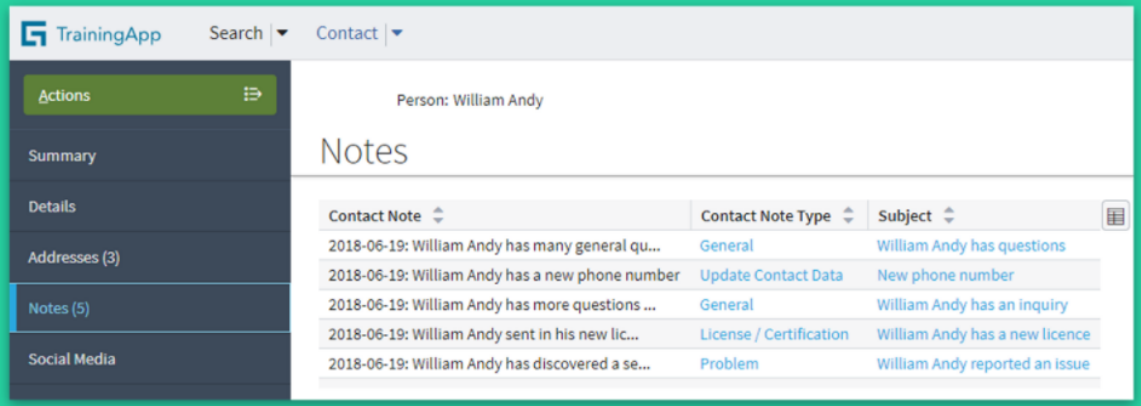
Notes

In the screenshot shown here, what is the expression to return the count of notes that are regarding a problem? What is the count value?

Menu

Answer

In the screenshot shown here, what is the expression to return the count of notes that are regarding a problem? What is the count value?




```
var countNotes = notes.countWhere( \ note ->
note.ContactNoteType == typekey.ContactNoteType.TC_PROBLEM)
```

The total number of elements in the ContactNotes array is 1.

Education

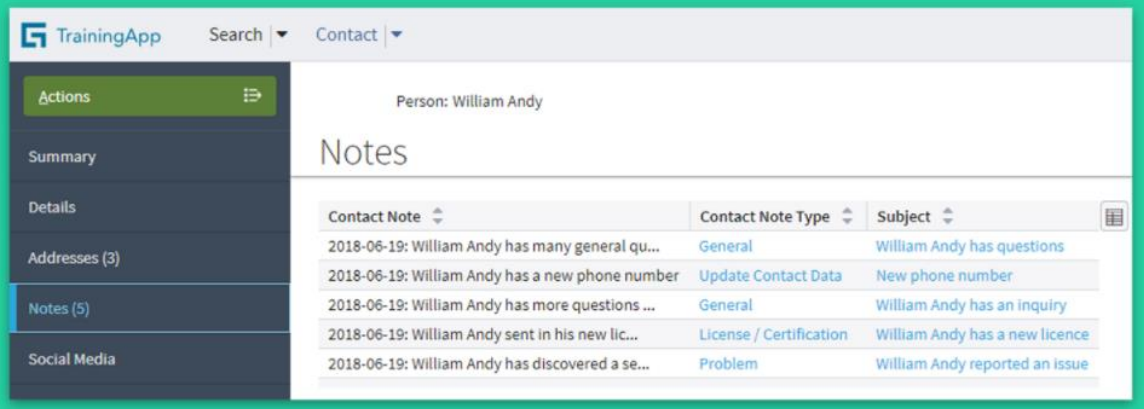
Notes

In the screenshot shown here, what is an expression to return an array of ContactNotes where the note was created after January 31, 2014?

Menu

Answer

In the screenshot shown here, what is an expression to return an array of ContactNotes where the note was created after January 31, 2014?




```
var filteredNotes = notes.where( \  note -> note.CreateTime >=
"02/01/2014".toDate() )
```

Education

PREV NEXT

GUIDEWIRE

Notes

Gosu Rules

Menu

Gosu Rules

Education

PREV NEXT

Notes

Gosu rules

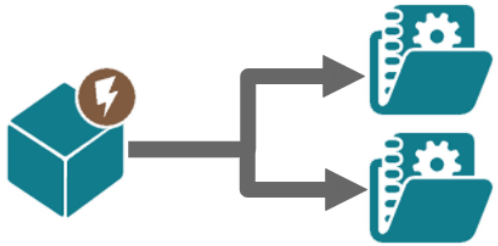
Note: Guidewire strongly recommends that you develop and document the functional logic of rules before attempting to turn that logic into rules within an InsuranceSuite product. Gosu rules require in-depth domain knowledge and technical expertise to create. After you make changes to Gosu rules, you typically need to restart the application server.

Menu

Gosu rules

- A **Rule** is a Gosu class with the file extension .gr
- A Rule is a single decision in the following form:

```
If {some conditions}
Then {take some actions}
```



- Example:
  - If ABPerson doesn't have an email address, then create a Flag Entry to notify the TrainingApp users
- When an Event happens to an Entity, different set of Rules might be executed by the Guidewire application
  - Example: when ABContact is saved the system runs
    - The ABContactPreupdate Rules and
    - The ABContactValidation Rules

Education

Notes

Examples in Guidewire applications

Shown here are various examples of Gosu rules in use in InsuranceSuite applications.

Menu

Examples in Guidewire applications

- Activity escalation rules
  - Escalate activity that is open for too long
  - Triggered when activity is open past its escalation date

- Claim assignment rules
  - Assign given claim to a group and user
  - Triggered when claim is created or needs reassignment

- Account validation rules
  - Verify that account is valid before committing it to database
  - Triggered when account is created or modified

Education

Notes

Gosu rule anatomy

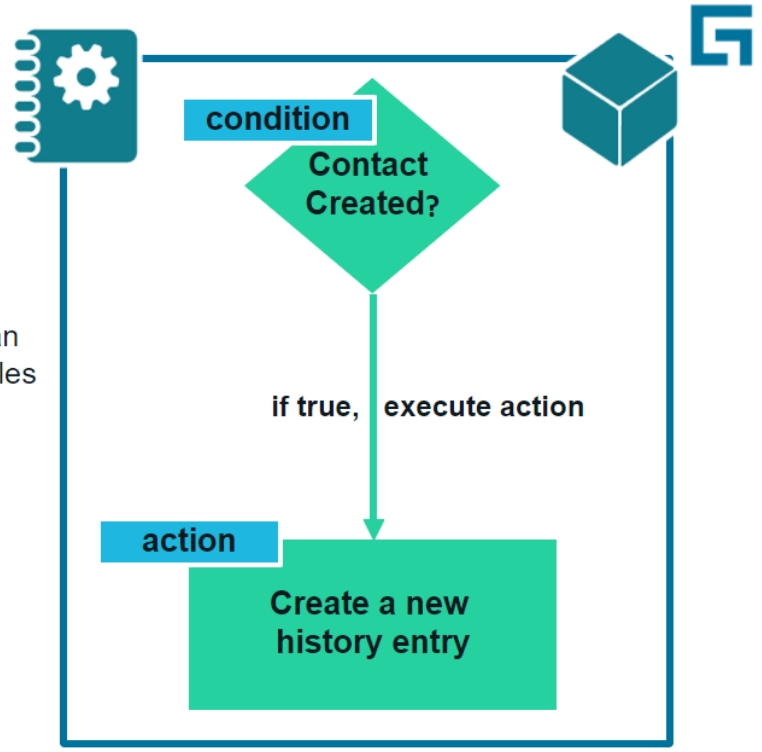
Setting a rule condition to "true" means that the action will always be performed whenever the rule is evaluated.

Notice that there is no "if" in the condition. This is assumed. The only valid condition is a boolean expression.

Menu

Gosu rule anatomy

- **Root entity**
  - Input parameter for the rule
- **Name**
  - Unique and follows naming convention
  - Example: **ABPU1010 – Contact Created** is an example of a naming convention for Gosu Rules decided on by the development team
- **Condition**
  - Expression that evaluates to true or false
- **Action**
  - If condition is true, the Guidewire application executes the action
  - If condition is false, nothing happens



Education

PREV NEXT

GUIDEWIRE

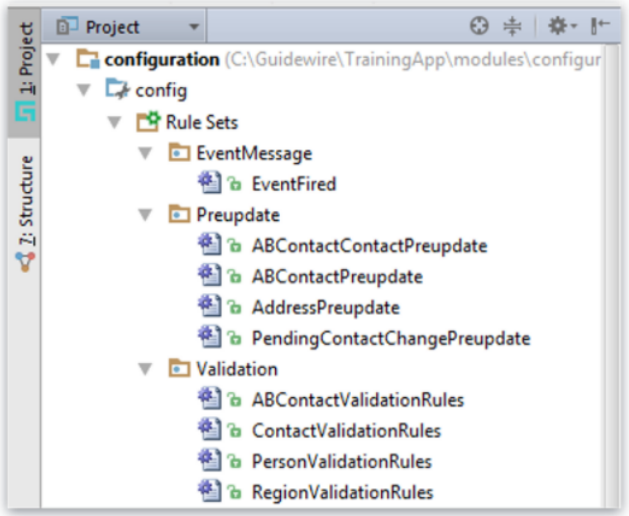
Notes

Hierarchy: Rule Set Category (1 of 2)

You can create new rule set categories. This is recommended only in the uncommon situation of an implementation that needs a new type of rule set category that is completely unlike any existing rule set category, however. To create a new rule set category, right-click the Rule Sets node and select New > Rule Set Category. You can also create a rule set category as a child node of an existing rule set category if there is a need to subgroup rule sets.

Hierarchy: Rule Set Category (1 of 2)

- A **Rule Set Category** is a collection of Rule Sets that have the following 2 things in common:
  - High-level business purpose
  - Trigger (Event)



Education

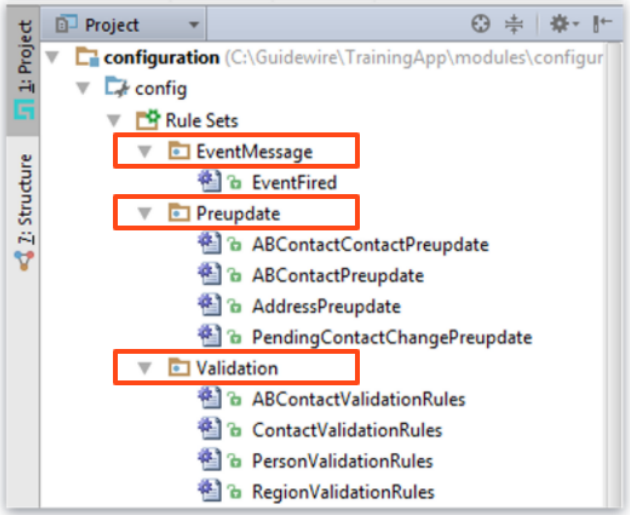
Notes

Hierarchy: Rule Set Category (2 of 2)

Guidewire applications contain the triggers for event messaging, preupdate, and validation. These triggers are associated with their respective rule set categories. This is true of all Guidewire applications and not just TrainingApp. If you create a new rule set category, you must also write the code necessary to trigger the rule sets in the rule set category. For more information, consult the Rules Guide in documentation.

Menu

Hierarchy: Rule Set Category (2 of 2)



In Project View, the top-level node is Rule Sets, a misnomer. The physical folder is …\config\rules\.

- Examples:
  - EventMessage
    - Performs event processing and generates messages about events that have occurred
    - Triggered during save
  - Preupdate
    - Performs domain logic or validation that must come before committing entity
    - Triggered during save
  - Validation
    - Performs validation that must come before committing entity that ensures data is valid
    - Triggered during save

Education

Notes

Hierarchy: Rule Sets

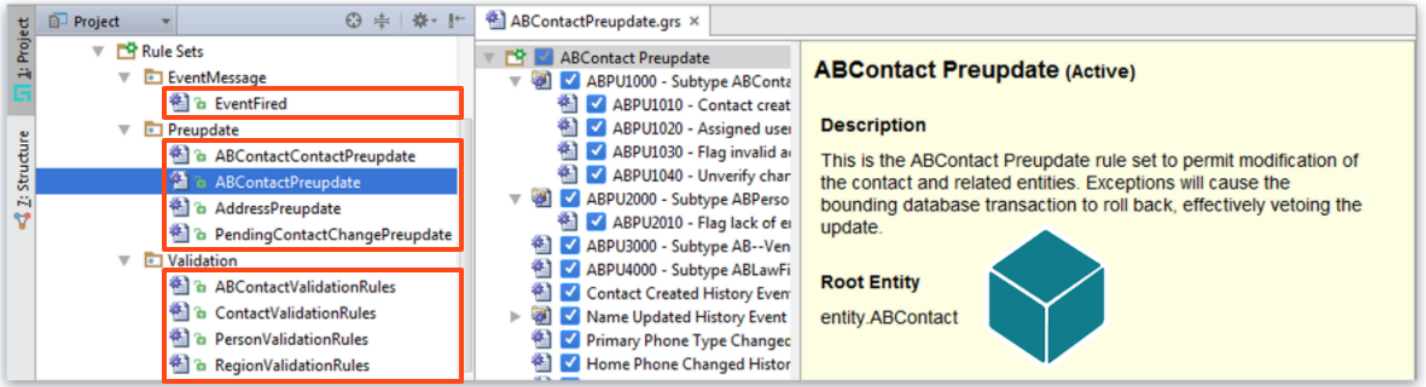
You can think of a rule set as a logical grouping of rules that are specific to a business function. You organize the rules in a rule set into a hierarchy that fits your business model. In other words, a rule set combines many individual rules into a useful set to consider as a group. One way that Guidewire helps you define the organization of rules is with a root entity and common triggers. A rule set is a collection of rules that share the same root entity and share common triggers. You write code for new rule set categories to define the triggers associated with a rule set. If you create rule sets for an existing rule set category (Event Messaging, Pre-update, and Validation), you do not need write trigger code.

Execution of rules always occurs at the rule set level. Unless the rules engine encounters an exit() command, all rules in the rule set are executed.

TrainingApp is built by heavily customizing an instance of ContactManager, and therefore has the same rule sets as ContactManager. These include:

- The Event Fired rule set, which is associated with MessageContext and triggers when a MessageContext object

Hierarchy: Rule Sets



- A **Rule Set** combines many individual rules into a useful set to consider as a group
- A Rule Set defines the root entity type
  - All the Rules in the set will be attached to same the triggering entity (root entity)
- Only the Rule Set Editor exposes individual rules.

Education

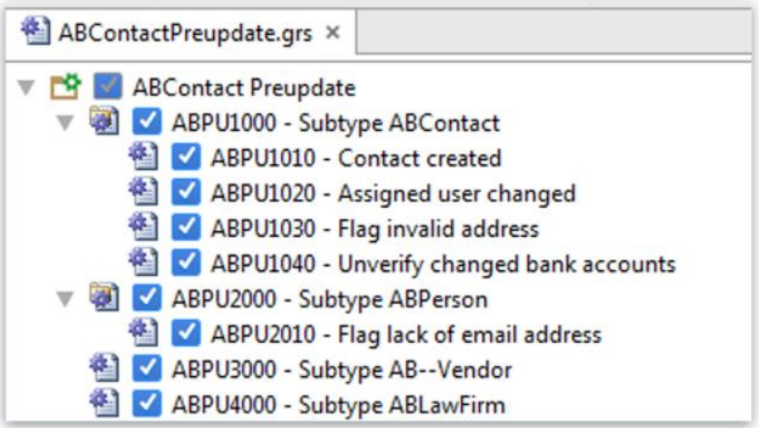
Notes

Hierarchy: Rules

If a rule has child rules, but the parent rule condition is false, neither the parent action nor the child rules are executed. In the slide example, the active (checked check box) rules are executed in the following hierarchy:

- ABPU1000 gathers together the rules relevant for all contacts. The child rules are executed only if this condition is true.
  - ABPU1010 creates a new history event when a contact is created.
  - ABPU1020 executes the necessary actions when a contact's assigned user changes (such as creating a note to record the change).
  - ABPU1030 executes the necessary actions when a flagged contact is unflagged (such as creating a note to record who unflagged the contact).
- ABPU2000 checks to see if the contact is an ABPerson. The child rules are executed only if this condition is true.
  - ABPU2010 flags any person who does not have an email address.

Hierarchy: Rules



- A **Gosu Rule** consists of a root entity, an expression that resolves to true or false, and an action that is executed if the condition is true
- Rules can have child rules
- If parent condition is true, Guidewire executes action and then executes all child rules

Education

Notes

What is the significance of a rule set's root entity?

Menu

Answer

What is the significance of a rule set's root entity?

Every rule set is tied to an entity, known as the root entity. This determines which class of objects the rule set is tied to. It also determines the object available to the rules in the rule set when the rule set is triggered.

Education

Notes

Gosu enhancements

Menu

Gosu enhancements

Education

PREV NEXT

Notes

Gosu enhancements

Guidewire types that can be enhanced include entities, Java classes, permissions, and SOAP entities. A fundamental reason for enhancing an entity is to add behaviors to a type that would not be appropriate to do by creating a subtype.

Menu

Gosu enhancements

- Gosu enhancements allow you to augment classes and other types with additional concrete methods and properties
- The most valuable and useful uses of this feature are:
  - to add new properties and methods to Guidewire Entities. Entities are defined in XML files and source code is automatically generated, thus cannot be modified, and
  - to define additional utility methods on a Java class or interface that cannot be directly modified. For example, the source code is unavailable, or a given class is final (cannot be subclassed)
- If an Entity/Java class is enhanced, then all the subtypes/subclasses will automatically inherit all elements from the enhancement

Education

Notes

Enhancement components

Enhancements can have any number of the following: getters, setters, and methods.

Menu

Enhancement components

Getter properties	Setter properties	Functions
• Calculate derived values	• Take input parameters, and/or	• Set field values that require additional logic • Modify other objects, and/or • Create objects

Education

Notes

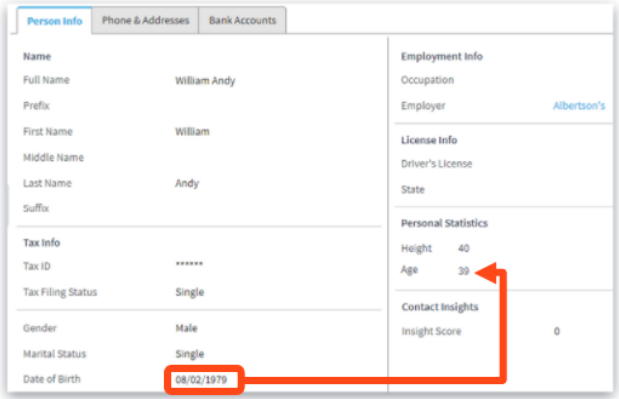
Derived (virtual) properties

In the slide example, ABPersonEnhancement.gsx is the enhancement file that extends the ABPerson entity with an Age property.

Menu

Derived (virtual) properties

- ABPersonEnhancement.gsx enhances ABPerson entity and defines the Age getter property
- Property calculates age based on the date of birth value
- Derived values should NOT be stored in the database, because storing them would be redundant
  - They are calculated from physical fields



Education

Page 6

PREV NEXT

Notes

Getter property

A getter is appropriate when you can derive a value and the derived value does not need to be stored in the database. In the slide example, ABPersonEnhancement.gsx is the enhancement file that extends the ABPerson entity with an Age property.

Menu

Getter property

- A **getter property** is used to calculate a derived value
  - Property not declared at Data Model level
    - As a result the value is not stored in database
  - Code cannot receive input parameters
  - Code can only return a value and should not alter any data
  - Null safe
- Example: `ABPerson.Age`
  - Derives value by calculating number of years between date of birth and current date
  - Returns "Unknown" if date of birth is null
  - Storing Age in database is redundant

Education

Notes

Setter property

An implementation of any Guidewire application will have a small number of setters because of the limiting criteria for use. A setter is appropriate only when you need to set one or more fields based on a single input value and—if there is a single field to be set—it is a field that cannot be set by simply assigning the input value to the field.

Possible use cases for setters could include:

- Adding a value to one field out of several possible fields (such as adding a new phone number to either Work Phone, Home Phone, or Cell Phone).
- Adding an object to an array and making that value the primary object (such as adding a new employee to an ABCompany and making that employee the primary contact).
- Adding a value to a field that must be converted to a different unit (such as receiving a dollar amount in Euros but needing to store that amount as dollars).
- Converting an integer value to a typekey value (such as receiving an integer that represents the number of employees and using that to set a CompanySize typekey value which could be: Under 100, 101 to 500,

Setter property

- A **setter property** takes a single input value and uses it to modify the associated object
  - Property not declared at data model level
  - Value given to setter may or may not be stored in database
  - Value manipulated by setter is typically stored in database
  - Code must receive exactly one input parameter
  - Should not be used to alter other objects
- Example: `ABPolicyPerson.HeightInInches(arg)`
  - Getter takes height from database (in meters) and converts it to inches (so, for example, it can be displayed in UI)
  - Setter takes height (in inches) and converts it to meters before saving it to database

Education

Notes

Gosu enhancements for Java

Gosu enhancements provide additional methods (functionality) on a Guidewire entity. For example, suppose that you create an enhancement to the Activity entity. Within this enhancement, you add methods that support new functionality. Then, if you type Activity. (Activity dot) within any Gosu code, Studio automatically uses code completion on the Activity entity. It also automatically displays any methods that you have defined in your Activity enhancement, along with the native Activity entity methods.

Gosu enhancements for Java

- Write Gosu enhancements for Java types
- Useful when creating generic methods that…
  - Do NOT relate to a specific entity
  - Extend the functionality of a base data type
- Examples:
  - GWBaseIntegerEnhancement enhances `java.lang.Integer`
  - GWBaseDateEnhancement enhances `java.util.Date`
  - GWBaseListEnhancement enhances `java.util.List`
- Java enhancements typically contain static functions
- Deploy and debug the same way as other enhancements

Education

Notes

What type of logic does a getter implement? A setter? A method?

Menu

Answer

What type of logic does a getter implement? A setter? A method?

The correct responses are:

Getter: Logic that returns a derived value and does not take parameters or change other data.

Setter: Logic that takes a single input value and uses that to modify some other field or set of fields on the given object.

Method: Logic that requires parameters, changes data, or otherwise does more than simply deriving or setting a value.

PREV NEXT

GUIDEWIRE

Notes

About Pattern Matching

This section discusses pattern matching.

Menu

About Pattern Matching

Education

PREV NEXT

Notes

Pattern matching

Validation is a general application behavior that prevents a user from saving invalid business data. Field-level validation is a validation behavior tied to one or more specific data fields.

A regular expression defines a pattern. The save is not allowed if the data entered by the user doesn't match the pattern.

Input masks both display a watermark in the field and guide the user to input the correct format of data. An input mask does not restrict data commit, but it does guide the user to input the data in a certain format.

Pattern matching

- Pattern matching is a field-level validation technique
- Verifies that the user entered the data in the correct format for that field
  - E.g.: tax id must be a nine-digit number with hyphens after the third and fifth digits
  - Prevents the user from saving invalid business data
- Could be implemented in the data model or the UI tier
- Main components:
  - Regular expression
  - Input mask (optional)

Notes

Pattern matching - regular expressions

The pattern for the Email validator is:

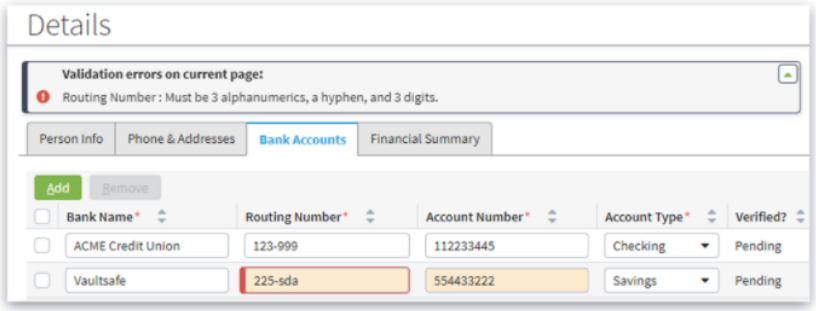
- Any non-empty string, specified as: .+
- An "at" symbol, specified as: @
- Any non-empty string, specified as: .+
- A period, specified as: \.
- Any non-empty string, specified as: .+

Characters that are not treated as literals and must be escaped include: [ ] ( ) { } . * + ?

For a complete listing of syntax for validator patterns, consult the Configuration Guide for your product.

Pattern matching - regular expressions

- A regular expression defines an abstract pattern
- Examples:
  - Email address: `.+@.+\..+`
  - Routing number: `[0-9a-zA-Z]{3}-[0-9]{3}`



Education

Notes

Pattern matching – input masks

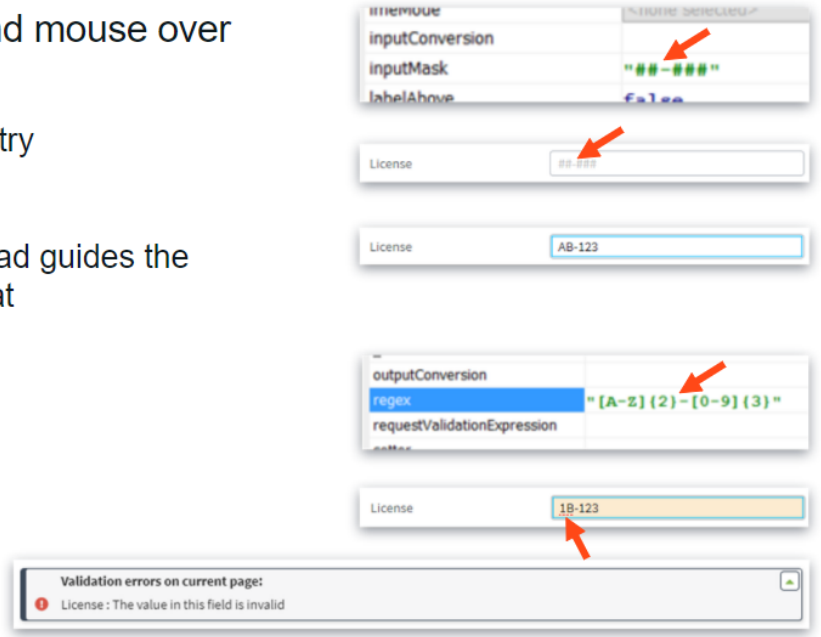
Input masks and regular expressions guide the user input for the field. Client-side regular expressions notify the user of an issue with the field input when a placeholder character (#) is used in the inputMask expression. An input mask identifies to the user the specific form of the data to enter into a given field. Typically, an input mask shows a basic pattern for a field. Input masks can also guide the user to input a specific character input in the field.

When an input mask contains placeholders (e.g., #) but does not specify either a regex property or does not have an associated entity field validator, the framework generates an implicit regular expression based on the inputMask property definition. As the user types in the field, the implicit regex triggers client-side validation, but does not restrict data commit. Instead, it guides the user to input the data in a certain format. In the example above, the field value should be 5 characters in length, the first two being upper case letters and the last three being numbers.

It should be noted that in addition to placeholders (#) in an inputMask property expression, you can force certain characters to be in the data. For example, "AB-###" forces the field to have the upper-case letters A and B to be the first two characters in the entry.

Pattern matching – input masks

- Appears as a watermark in field and mouse over tooltip
  - Changes to a darker color with data entry
  - No format warning
  - Does not restrict data commit but instead guides the user to input the data in a certain format



Education

PREV NEXT

Notes

Pattern matching- practices comparison

Shown here are the pros and cons and pattern matching in the Data Model and the User Interface.

Menu

Pattern matching- practices comparison

Data model

- Pros
  - Application enforces compliance in the UI and through API
  - Supports full localization
  - Robust extensibility
- Cons
  - Static values only: regex and input mask cannot change based on business logic
- Implemented as a reusable field validator

User interface

- Pros
  - Dynamic values: regex and input mask can change based on business logic
  - Can restrict data commit
- Cons
  - Only a few widget support it
  - Must be configured per widget
  - No custom error message option
  - Extensibility a big concern
- Implemented as widget properties

Education

PREV NEXT

Notes

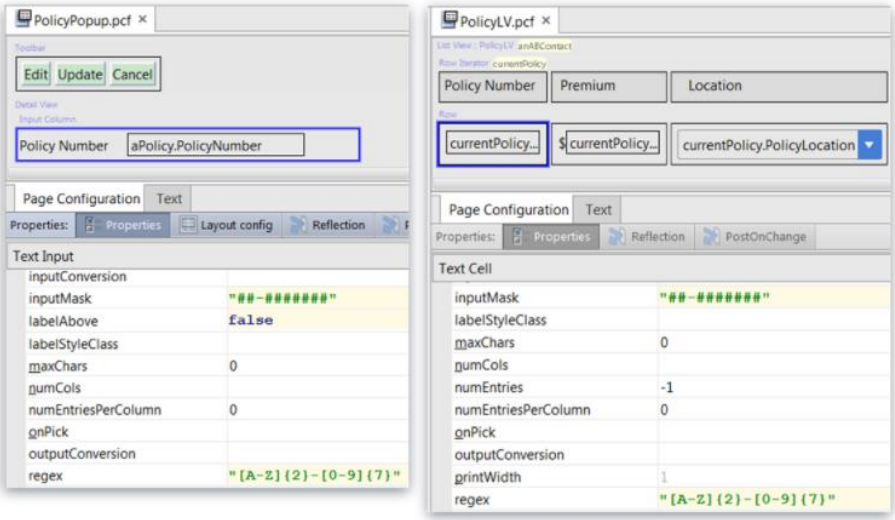
Example: pattern matching in the UI

PCF Format Reference: <application-installation-folder>/modules/pcf.html

Menu

Example: pattern matching in the UI

- Implementing the user story in the UI requires duplicating regex and input mask
- **inputMask** and **regex** properties can be dynamic
  - To improve the design, a function call could be used instead of hard-coding the strings
- No custom error message
- Not all the widgets have the **regex** and **inputMask** properties
  - Refer to PCF Format Reference

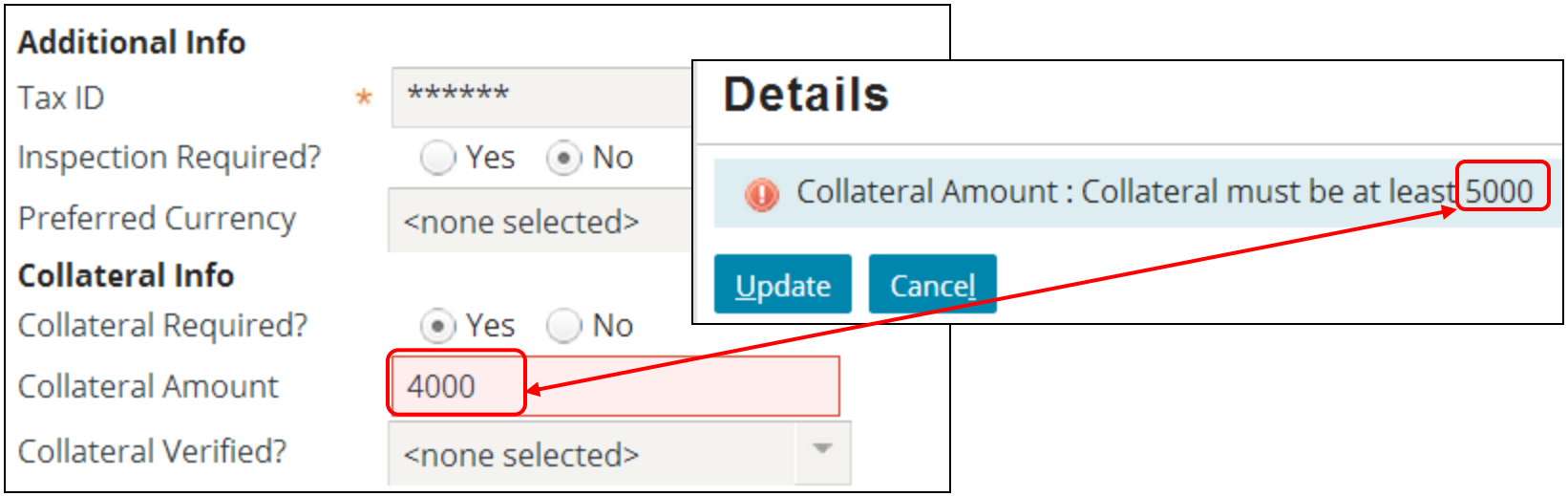


Education

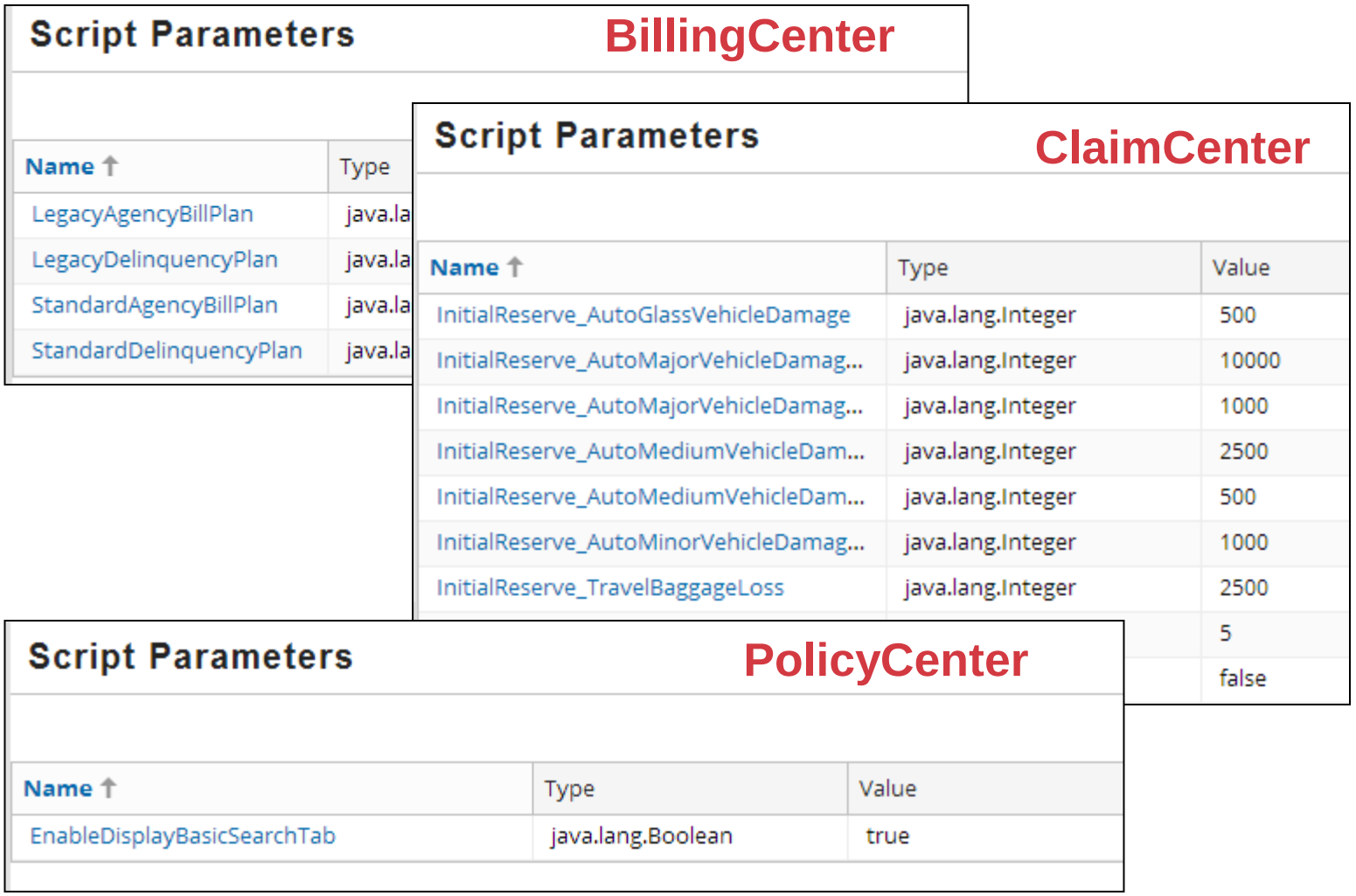
PREV NEXT

Script parameters

- A **script parameter** is an application-wide global constant
  - Value maintained by administrator
  - Can be referenced in any Gosu code
  - Typically used for values that may change over time and therefore should not be hard-coded



Examples from base applications



Example 2: Toggling application behavior

```
// This functions creates a history event identifying that the given
// contact was viewed by the current user. (This function does nothing
// if the RecordInHistory-UserViewsOfContacts script parameter is set to false.)

   if (ScriptParameters.RecordInHistory_UserViewsOfContacts) {
```

when set to true...

History

Date	Event Type	Description
09/19/2013	Viewed	Alice Applegate viewed this contact.
09/19/2013	Viewed	
09/18/2013	Viewed	
09/18/2013	Viewed	
09/16/2013	Viewed	
09/16/2013	Created	

when set to false...

Date	Event Type	Description
09/18/2013	Viewed	Super User viewed this contact.
09/16/2013	Viewed	Super User viewed this contact.
09/16/2013	Viewed	Super User viewed this contact.
09/16/2013	Viewed	Super User viewed this contact.
09/16/2013	Created	Contact William Andy created

Notes

About Partial Page Update

This section discusses partial page update.

Menu

About Partial Page Update

Education

PREV NEXT

Notes

Partial page update: Layout re-render

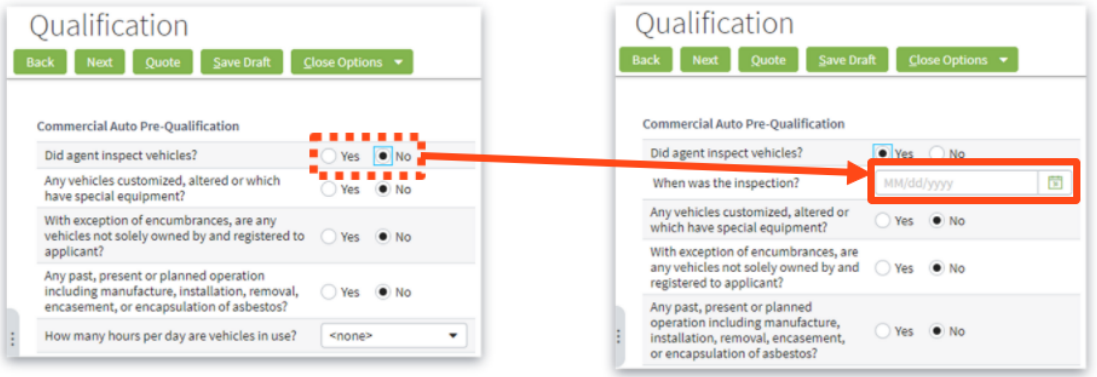
In the slide example, the Company Info card has both an Inspection Required field and Inspection Date field. However, the Inspection Date field is visible only if the Inspection Required? field is set to Yes.

The InspectionRequired widget displays a Yes or No radial buttons. A selected Yes radial button equates to true and a selected No radial button equates to false. A null value is represented as neither radial button selected.

Dynamic widget behavior differs from a property that evaluates an expression in that the user changes the business data while editing it and that change affects the behavior of the widget. In the previous examples, the changes are present only after the data has been committed.

Partial page update: Layout re-render

- Responds to a user changing business data **while** it happens and changes the layout of widgets on the screen
- Example:
  - While a user changes the Did agent inspect vehicles? field to Yes, a secondary question and the Date Input widget becomes visible



Education

PREV NEXT

Notes

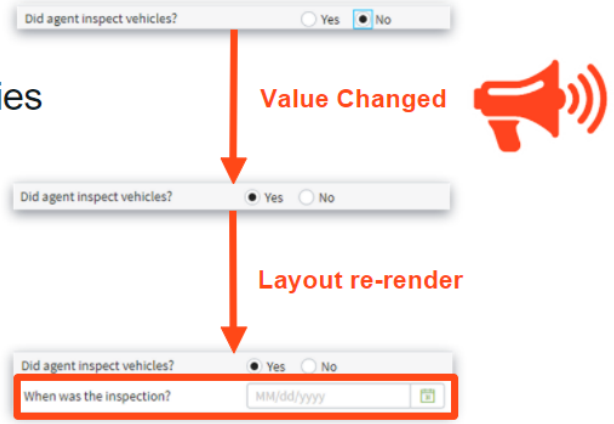
Partial page update: Layout re-render (Continued)

There are two general and non-exclusive categories for a partial page update: layout re-render and DATA_ONLY.

In the layout re-render example above, whenever the value of the InspectionRequired field is changed, all user editable data is sent from the client to the application server. No data is committed. The application server processes the request and returns instructions and user editable data to the client. The client re-renders the layout that shows an Inspection Date field.

Partial page update: Layout re-render (Continued)

- Layout re-rendered for the page
- No data committed
- Applies to following dynamic widget properties
  - visible
  - editable
  - available
  - required



Education

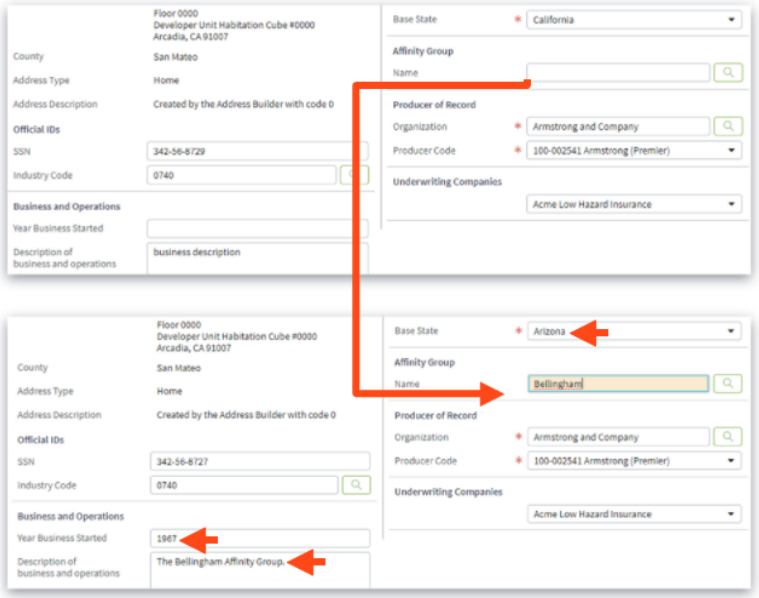
Notes

Partial page update: DATA_ONLY

In the DATA_ONLY example, the widget configuration enables dynamic widget behavior for the Affinity Group name TextInput widget. When a user changes the value of the Name field, the client makes a request to the application server. The application server processes the request and returns instructions to the client. In this case, the instructions tell the client to render only the newly supplied data for the Business and Operations fields (year started and description of business)

Partial page update: DATA_ONLY

- Responds to a user changing business data **while** it happens
  - Not after the user navigates to the page
  - Not after the user commits or tries to commit the data
- Example:
  - While a user changes the Affinity Group Name field, values for other fields are rendered with changes (read-only in UI)



Education

Notes

Two kinds of widget properties

In the slide example, the id property of a widget is a static property. The widget always has the same ID, regardless of the state of the application or the values of any business data. The property does not evaluate an expression, just the value of a type, such as a String or Integer value.

The editable property evaluates a boolean expression and the expression can be a dynamic value, in this case, one that returns true or false. The label property evaluates a string expression. This lesson discusses display keys later on. Display keys are a type of string expression in Guidewire applications for localizing text. The value property is also a dynamic property. The value property evaluates an object expression that binds the object property to it.

Both types of widget properties can be required properties for a specific type of widget.

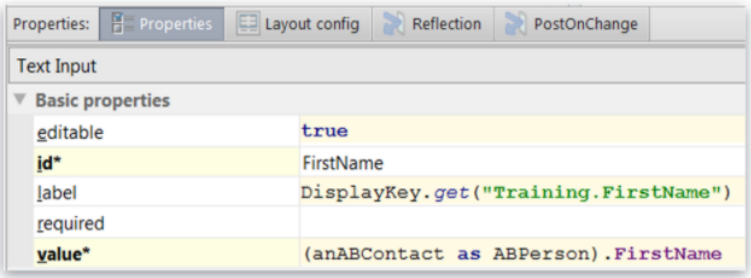
Two kinds of widget properties

Static Property

- Evaluates a static value that is immutable, never changes
  - id requires a static value

Dynamic Property

- After a user navigates to a page or clicks update, evaluates an expression and returns a value
  - editable requires boolean expression
  - label returns a string expression
  - value returns an object expression



Education

Notes

Dynamic properties evaluate expressions

Not all widgets have all properties. Refer to the PCF Format Reference for more examples. Recall that the PCF Format Reference defines the properties for every widget, including the type of value the property takes. Note that the PCF Format Reference refers to widget properties as "attributes". To open the PCF Format Reference, open <ApplicationRootDirectory>\modules\pcf.htm file in your web browser.

You can also consider the value property of a widget to be a dynamic property because it evaluates an object expression.

Visible is typically not set to false. It is either set to true or it is set to an expression that renders the widget visible or not visible. The default value for the visible property is true.

Dynamic properties evaluate expressions

- Available
  - Boolean expression which, if false, grays out the widget and its children
- Editable
  - Boolean expression which, if false, makes the widget and its children read-only
- Required
  - Boolean expression which, if true, then a value must be filled out by the user
- Visible
  - Boolean expression which, if false, completely hides the widget and its children

Notes

Post On Change properties

In certain cases, you may want to toggle (disable or enable) targeted Post On Change behavior for a specific widget.

In the slide example, the Invoicing Method field will disable Post On Change if the currency is not USD.

For all other currencies, the Post On Change will be disabled. The widget will not announce a change in its value; data will not make a round-trip between client and the application server; the page will not be re-rendered and data won't be refreshed.

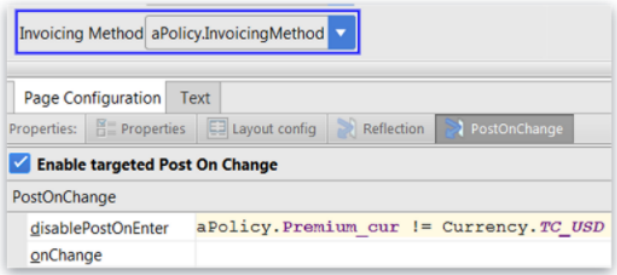
In the slide example, the Gosu expression defined for the onChange property invokes the setFieldsOnResolution() function. The function is defined in the entity enhancement, FlagEntryEnhancement.gsx.

```
16 /* This function is called when
a FlagEntry's resolution field
17 is set. This function sets the
UnflagDate and UnflagUser
18 fields. This serves the role of a
"FlagEntry Pre-Update"
19 rule set.
20 */
21 function
setFieldsOnResolution() : void {
22 this.UnflagDate =
gw.api.util.DateUtil.currentDate()
23 this.UnflagUser =
User.util.getCurrentUser()
24 } // end of function
```

Post On Change properties

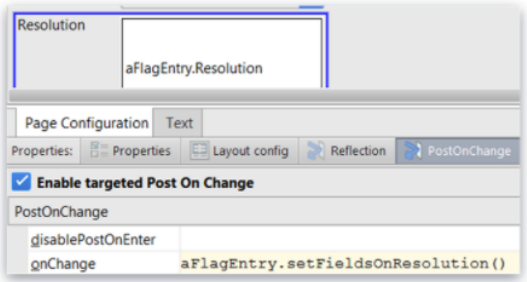
disablePostOnEnter

- If evaluated true when page is rendered, this field won't trigger Post On Change
- Default is false



onChange

- Defines a Gosu expression to invoke when user changes the value of the widget
- Causes immediate post back to the server



Education

Notes

Target property (earlier versions)

When specified as a widget, the target area can be a specific widget or a grouping of widgets. For example, if the Widget ID corresponds to an Input Set, then the widgets in the input set are re-rendered.

There is an additional layout re-render configuration for backwards compatibility: no value. The no value configuration consists of enabling targeted post on change alone with no property configurations. The backwards compatibility configuration has no property values specified and causes the re-rendering of the entire page and refreshes all user-editable data. There is a high-performance cost for implementing targeted Post On Change without specifying a target and any other properties.

In current InsuranceSuite applications, targeted update is automatic for any widget where Targeted Post On Change is enabled, and the target property is redundant.

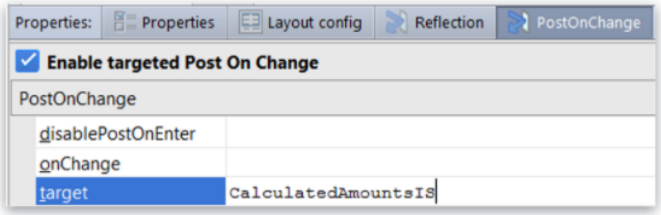
Target property (earlier versions)

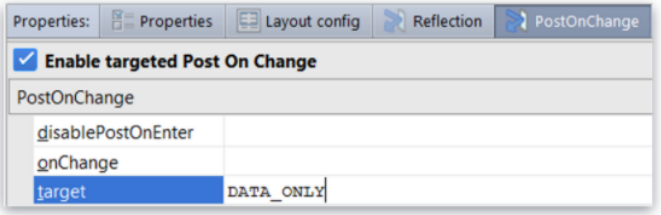
Layout re-render

- Widget ID
- Target area re-rendered
- All editable data refreshed
- Widgets can also be grouped and parent container can be targeted

Data update

- DATA_ONLY
- No layout re-render
- All editable data refreshed
- Better performance





Education

Notes

Name two properties that evaluate expressions.

Menu

Answer

Name two properties that evaluate expressions.

Possible answers include:

- editable
- required
- visible
- available

Education

PREV NEXT

Notes

How is a partial page update implemented?

Menu

Answer

How is a partial page update implemented?

A partial page update is implemented by configuring dynamic widget behavior by enabling targeted post on change. Often the dynamic widget behavior configuration also includes configuring an expression for a widget property.

Education

PREV NEXT

Notes

What are the three properties that you can configure for a widget with targeted Post On Change enabled?

Menu

Answer

What are the three properties that you can configure for a widget with targeted Post On Change enabled?

disablePostOnEnter, onChange, and (with earlier InsuranceSuite versions) target.

Education

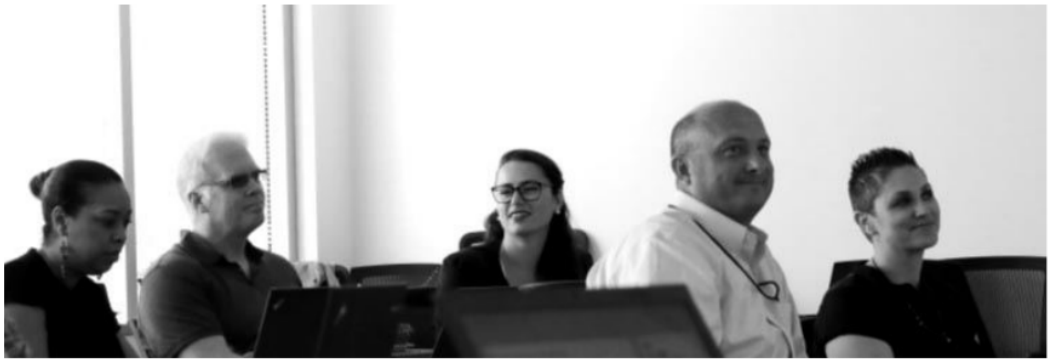
Notes

Introduction to Gosu

Menu

Education

Introduction to Gosu



NEXT ›

Notes

Using Gosu to configure the Application Tier

Menu

Using Gosu to configure the Application Tier

Education

PREV NEXT

Notes

Guidewire Gosu

Gosu is Guidewire's open-source, publicly available programming language. Gosu has elements of both procedural and object-oriented programming languages and is similar to JavaScript and Java.

Guidewire developed Gosu for several reasons.

First, there was a desire to have a single syntax that could be used to work with all of the elements relevant to Guidewire products (such as entities, display keys, classes, class enhancements, Java classes, permissions, and script parameters) even though these items have fundamentally distinct internal implementations. There was no existing language that provided this ability.

Second, there was a desire to have code auto-complete features in Studio. This is possible only with a statically typed language. Most scripting languages, such as JavaScript, Perl, Python, and Ruby, are dynamically typed.

In very early releases of InsuranceSuite, Gosu was known as GScript.

Guidewire Gosu

- Guidewire's programming language
  - Similar to Java and compatible with Java
  - Has elements of both procedural and object-oriented programming languages
- Executes fundamental application behavior
- Manages complex business processes
- Executes hierarchical Gosu Rules
- Specifies dynamic user interface behavior

Notes

Gosu in Guidewire applications (1 of 3)

A rule is a single decision in the following form of testing a condition and then taking an action: if {some conditions} then {take some action}. A rule set combines many individual rules into a useful set to consider as a group.

Gosu enhancements provide additional methods (functionality) on a Guidewire entity. For example, you can create an enhancement to the ABContact entity. Within enhancement, you add methods that support new functionality. Guidewire Studio will show the new method for an entity of the type ABContact.

Note: Enhancements are not limited to entities. Technically any Gosu or Java class can be enhanced.

Gosu in Guidewire applications (1 of 3)

Gosu Rules

- Apply specific programming logic for testing a condition and performing an action
- Examples:
  - Event Fired Rules
  - Pre-update Rules
  - Validation Rules

Entity Enhancements

- Extend entity functionality with programming logic as entity methods
- Examples:
  - ABCompany.maskTaxId() returns a value with a mask for the first 5 characters

Education

Notes

Gosu in Guidewire applications (2 of 3)

Shown here is a comparison between the use of Entity Names and Gosu Classes.

Menu

Gosu in Guidewire applications (2 of 3)

Entity Names

- Programming logic that defines how to display a name for an entity instance
- Examples:
  - Drop-down lists of contacts
  - DisplayName property of entity instance

Gosu Classes

- Encapsulate data and code for a specific purpose or function
- Examples:
  - Utility helper classes for logging or string functions
  - Plugins for integrations with other applications

Education

PREV NEXT

Notes

Gosu in Guidewire applications (3 of 3)

A workflow executes processes whose execution is time-based or is triggered by events in external applications or is both triggered and time-based.

Workflows are available in all three primary Guidewire applications. PolicyCenter uses workflows to manage policy transactions (such as renewals). BillingCenter uses workflows to manage account delinquency and agency bill cycles. Although the feature is available in ClaimCenter, the base configuration of ClaimCenter does not use workflows.

Gosu in Guidewire applications (3 of 3)

Workflows

- Run custom business processes asynchronously, optionally with multiple states that transition over time
- Examples:
  - Renewal and cancelation workflows

Notes

Gosu Scratchpad

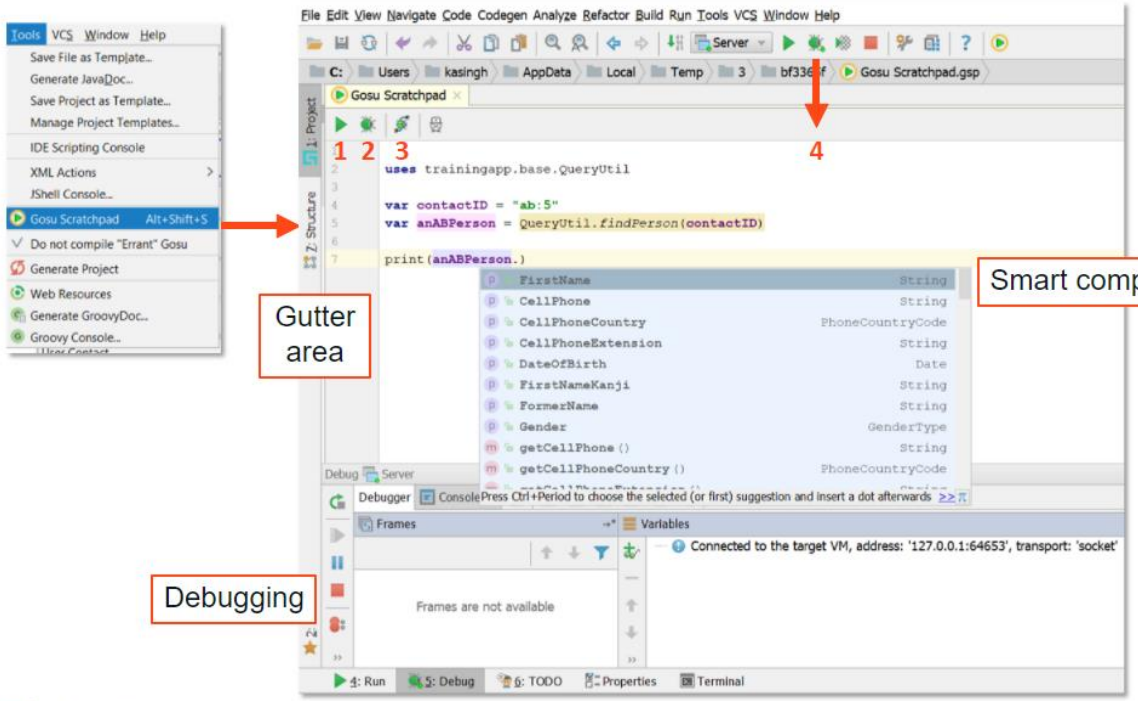
Gosu Scratchpad can be used to write, execute and debug Gosu code snippets.
To open up the Gosu scratchpad, select Tools and then Gosu Scratchpad
In the example above, we run a simple Gosu query to load data from the application database.
We can also execute transactions to commit test data.

The Gosu scratch pad supports common editor features

- Gutter Area
- Smart Completion
- Validation bar and
- Debugging

The Run in Debug Server process button (3) uses the Platform compiler while the Run (1) and Debug (2) buttons use the community (open source) compiler. There are a few minor differences between the two versions of Gosu, but those are out of scope for this course.
It is recommended to use the Run in Debug Process button (3) and use Platform compiler since all the applications are written using the Platform compiler.
The server must be running in debug mode (4) before you can run Gosu code using the debug server process button (3).

Gosu Scratchpad



1 Run

2 Debug

3 Run in Debug Server process

4 Run (Server) in debug mode

Education

Notes

Arrays

Configuration of Guidewire applications often focuses on working with data model entity arrays. An array defines a set of additional entities of the same type to associate with the main entity.

A related data model array is the associative array. An associative array provides a mapping between a set of keys and the values that the keys represent. A common example of this type of mapping is a telephone book, in which a name maps to a telephone number. Another common example is a dictionary, which maps terms to their definitions.

An array is a collection of data values, with each element of the array associated with a number or index.

In typical Gosu code, simply use angle brackets after the type name, such as String[] to represent an array of String objects. Use a zero-based index number to access an array member. If you create an array, you must explicitly define the size of the array or implicitly define the size by simultaneously defining the array elements. To access the elements of an array, you use an index expression.

Arrays

Data model entity array

- `<array/>` on parent entity to array entity
- `<foreignkey/>` on array entity to parent entity
- **Example**
  - ABContact defines FlagEntries array

Derived API entity array

- Entity enhancement or Guidewire API
- **Example**
  - ABContact AllAddresses property returns an Address array

Gosu datatype and object array

- A set of values are of the same type in a single collection
- **Examples**
  - `var a = new int[3]`
  - `var b = new int[] {1,2,3}`
  - `var c : int[] = {1,2,3}`

Education

Notes

Array methods that require logic

A variety of array methods are available in Gosu, including methods that get information about the array or that get members of the array that match a given condition.
For methods that require an argument, the argument must be an expression of a condition, such as returning *true* if any row in the array matches the condition, returning the first row matching the condition, or returning all rows that match the condition. Arguments must be in the form of a Gosu block expression.

Menu

Array methods that require logic

- Array methods include
  - Getting information about the array
  - Getting members of the array that match a given condition
- Some methods require an argument
- Argument must be an expression of a condition, such as:
  - Returns true if any row in the array matches the condition
  - Returns the first row that matches the condition
  - Returns all rows that match the condition
- Argument must be a Gosu block expression

PREV NEXT

Notes

Common array methods (1 of 2)

Common array methods include *hasMatch(condition)*, which determines if any element in an array matches the condition provided, and *countWhere(condition)*, which returns a count of elements in an array that match the condition.

Menu

Common array methods (1 of 2)

- `hasMatch(condition)`
  - Determine if any element in an array matches a given condition
  - Syntax: `array.hasMatch(\ name -> conditionToMatch)`
- `countWhere(condition)`
  - Returns count of elements that match a given condition in an array
  - Syntax: `array.countWhere(\ name -> conditionToMatch)`

PREV NEXT

Notes

Common array methods (2 of 2)

Additional array methods include *firstWhere(condition)*, which retrieves the first element of an array matching the condition, and *where(condition)*, which creates a target array consisting of the members of the source array that match the condition. For a more detailed description and information on other methods, see the product documentation.

Menu

Common array methods (2 of 2)

- `firstWhere(condition)`
  - Retrieve the first element in an array that matches a given condition
  - Syntax: `array.firstWhere(\ name -> conditionToMatch)`
- `where(condition)`
  - Retrieves a target array that consists of all the members of a source array that match a given condition
  - Syntax: `array.where(\ name -> conditionToMatch)`

PREV NEXT

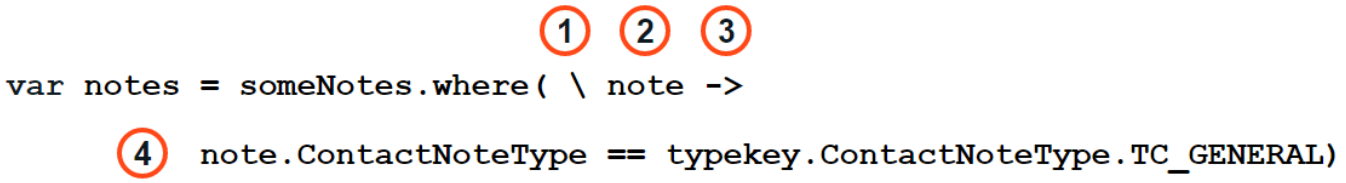
Notes

Blocks argument expression

Blocks are used in many situations where a method requires an expression as an input parameter, such as in arrays, queries, and transaction bundles for database transactions.
A block is an expression of logic passed to a method as an argument. Some array methods require conditions using a block.
The block consists of four parts:

1. \ , which identifies that the following is a block
2. An element name representing each element of the array
3. ->, which identifies the start of the condition
4. A condition, which relates to the element

Blocks argument expression



- A **block** is an expression of logic passed to a method as an argument
- Some array methods require conditions using a block
- The block consists of four parts:
  1. \ , which identifies that the following is a block
  2. An element name representing each element of the array
  3. ->, which identifies the start of the condition
  4. A condition, which relates to the element

Education

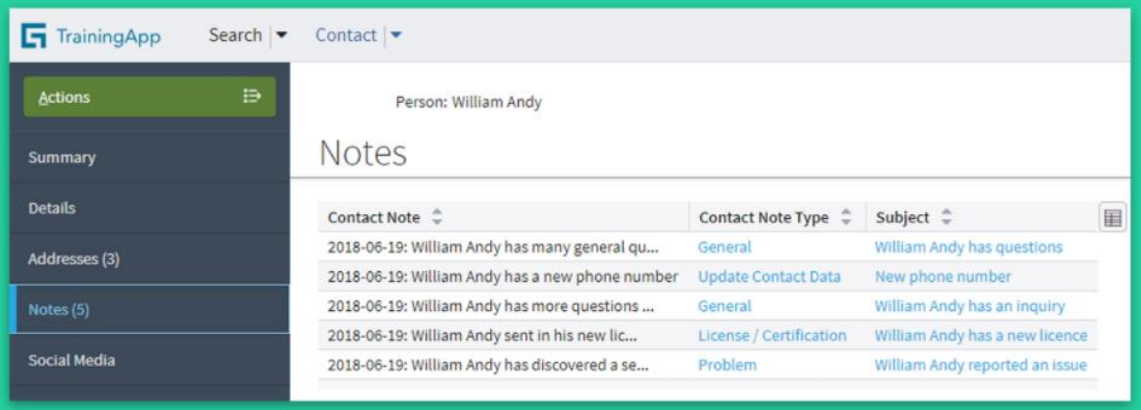
Notes

In the screenshot shown here, what is the array method to return the number of objects in the ContactNotes array? What is the value?

Menu

Answer

In the screenshot shown here, what is the array method to return the number of objects in the ContactNotes array? What is the value?




```
var count = array.length or array.Count.
```

The total number of elements in the ContactNotes array is 5.

Education

PREV NEXT

GUIDEWIRE

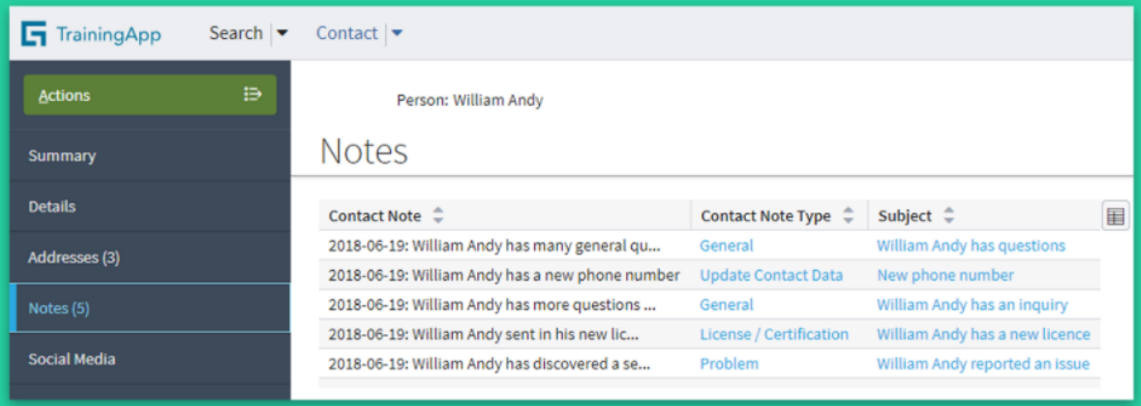
Notes

In the screenshot shown here, what is the expression to return the count of notes that are regarding a problem? What is the count value?

Menu

Answer

In the screenshot shown here, what is the expression to return the count of notes that are regarding a problem? What is the count value?




```
var countNotes = notes.countWhere( \ note ->
note.ContactNoteType == typekey.ContactNoteType.TC_PROBLEM)
```

The total number of elements in the ContactNotes array is 1.

Education

PREV NEXT

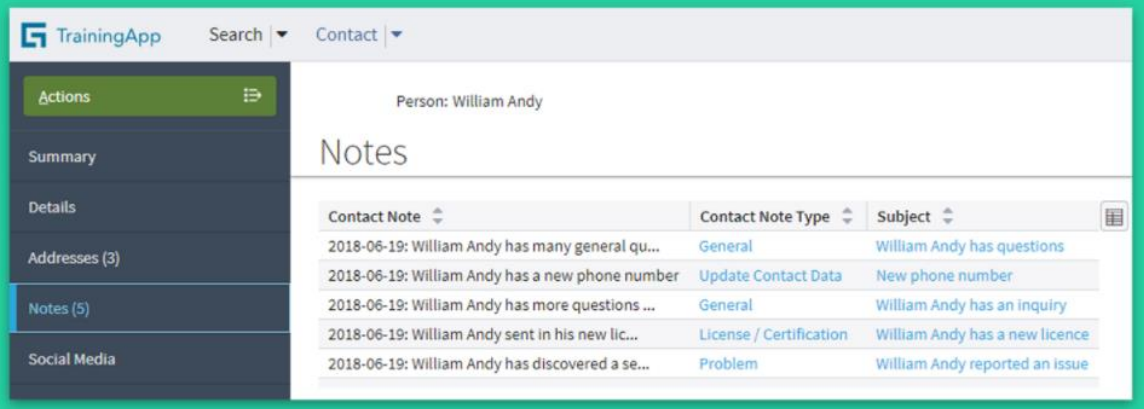
Notes

In the screenshot shown here, what is an expression to return an array of ContactNotes where the note was created after January 31, 2014?

Menu

Answer

In the screenshot shown here, what is an expression to return an array of ContactNotes where the note was created after January 31, 2014?




```
var filteredNotes = notes.where( \  note -> note.CreateTime >=
"02/01/2014".toDate() )
```

Education

Page 26

PREV NEXT

Notes

Gosu Rules

Menu

Gosu Rules

Education

PREV NEXT

Notes

Gosu rules

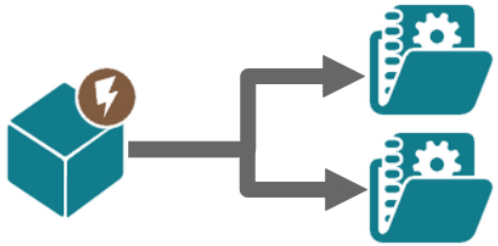
Note: Guidewire strongly recommends that you develop and document the functional logic of rules before attempting to turn that logic into rules within an InsuranceSuite product. Gosu rules require in-depth domain knowledge and technical expertise to create. After you make changes to Gosu rules, you typically need to restart the application server.

Menu

Gosu rules

- A **Rule** is a Gosu class with the file extension .gr
- A Rule is a single decision in the following form:

```
If {some conditions}
Then {take some actions}
```



- Example:
  - If ABPerson doesn't have an email address, then create a Flag Entry to notify the TrainingApp users
- When an Event happens to an Entity, different set of Rules might be executed by the Guidewire application
  - Example: when ABContact is saved the system runs
    - The ABContactPreupdate Rules and
    - The ABContactValidation Rules

Education

Notes

Examples in Guidewire applications

Shown here are various examples of Gosu rules in use in InsuranceSuite applications.

Menu

Examples in Guidewire applications

- Activity escalation rules
  - Escalate activity that is open for too long
  - Triggered when activity is open past its escalation date

- Claim assignment rules
  - Assign given claim to a group and user
  - Triggered when claim is created or needs reassignment

- Account validation rules
  - Verify that account is valid before committing it to database
  - Triggered when account is created or modified

Education

Notes

Gosu rule anatomy

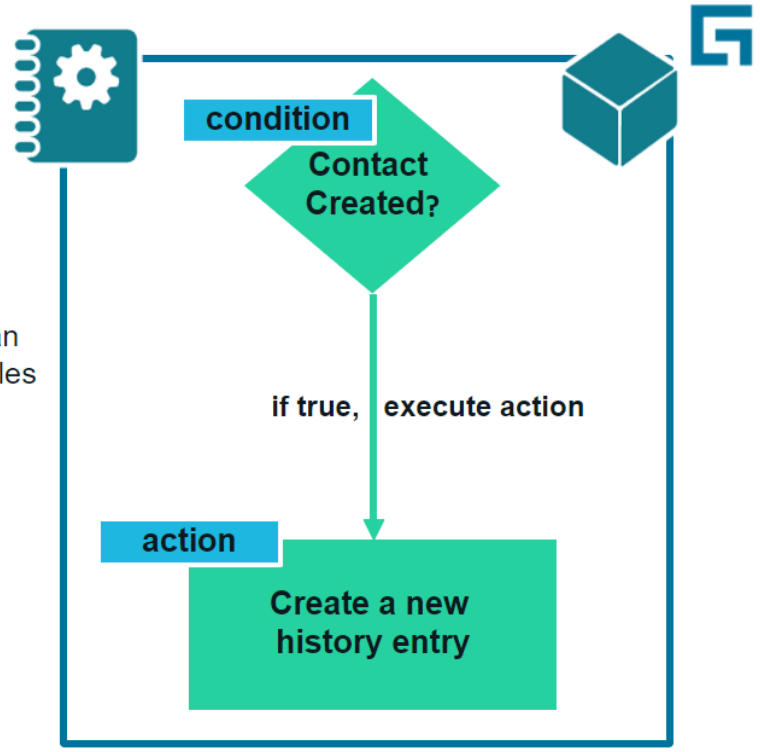
Setting a rule condition to "true" means that the action will always be performed whenever the rule is evaluated.

Notice that there is no "if" in the condition. This is assumed. The only valid condition is a boolean expression.

Menu

Gosu rule anatomy

- **Root entity**
  - Input parameter for the rule
- **Name**
  - Unique and follows naming convention
  - Example: **ABPU1010 – Contact Created** is an example of a naming convention for Gosu Rules decided on by the development team
- **Condition**
  - Expression that evaluates to true or false
- **Action**
  - If condition is true, the Guidewire application executes the action
  - If condition is false, nothing happens



Education

PREV NEXT

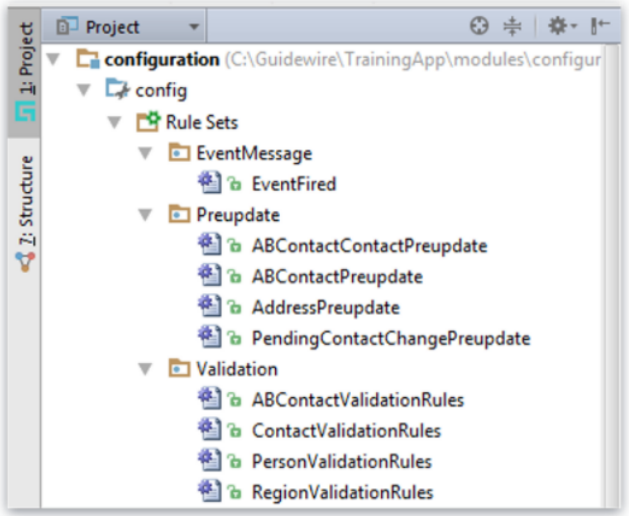
Notes

Hierarchy: Rule Set Category (1 of 2)

You can create new rule set categories. This is recommended only in the uncommon situation of an implementation that needs a new type of rule set category that is completely unlike any existing rule set category, however. To create a new rule set category, right-click the Rule Sets node and select New > Rule Set Category. You can also create a rule set category as a child node of an existing rule set category if there is a need to subgroup rule sets.

Hierarchy: Rule Set Category (1 of 2)

- A **Rule Set Category** is a collection of Rule Sets that have the following 2 things in common:
  - High-level business purpose
  - Trigger (Event)



Education

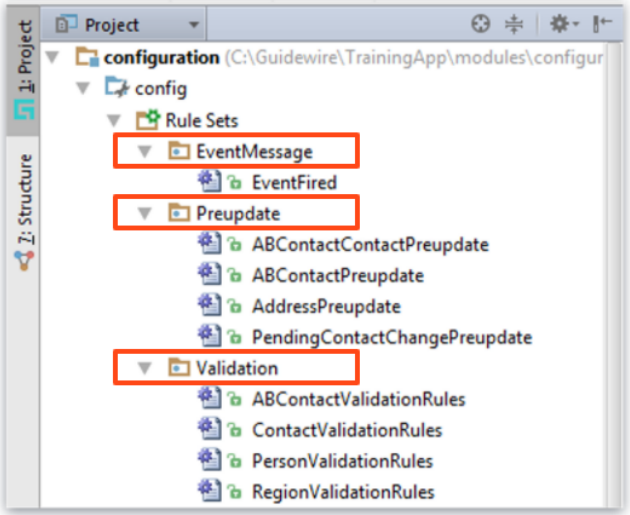
Notes

Hierarchy: Rule Set Category (2 of 2)

Guidewire applications contain the triggers for event messaging, preupdate, and validation. These triggers are associated with their respective rule set categories. This is true of all Guidewire applications and not just TrainingApp. If you create a new rule set category, you must also write the code necessary to trigger the rule sets in the rule set category. For more information, consult the Rules Guide in documentation.

Menu

Hierarchy: Rule Set Category (2 of 2)



In Project View, the top-level node is Rule Sets, a misnomer. The physical folder is …\config\rules\.

- Examples:
  - EventMessage
    - Performs event processing and generates messages about events that have occurred
    - Triggered during save
  - Preupdate
    - Performs domain logic or validation that must come before committing entity
    - Triggered during save
  - Validation
    - Performs validation that must come before committing entity that ensures data is valid
    - Triggered during save

Education

Notes

Hierarchy: Rule Sets

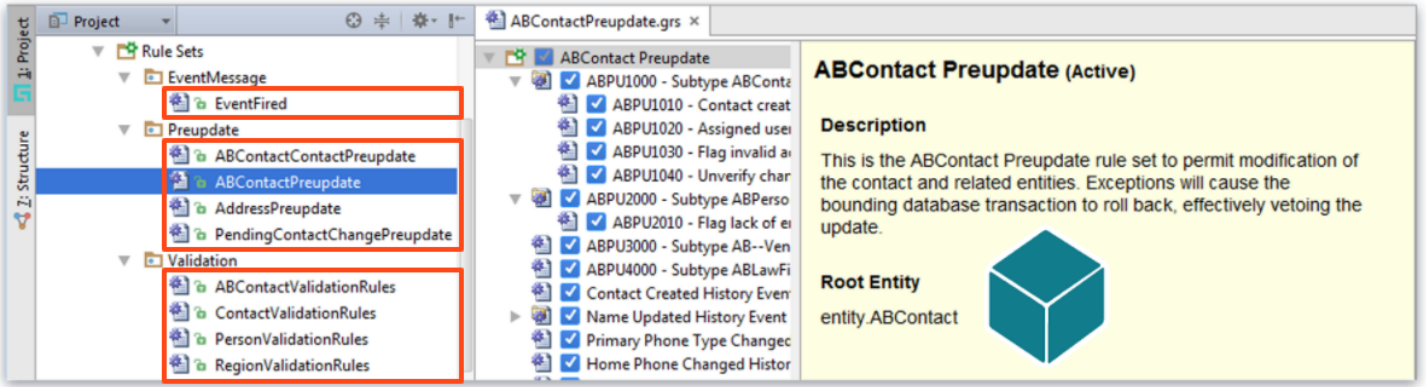
You can think of a rule set as a logical grouping of rules that are specific to a business function. You organize the rules in a rule set into a hierarchy that fits your business model. In other words, a rule set combines many individual rules into a useful set to consider as a group. One way that Guidewire helps you define the organization of rules is with a root entity and common triggers. A rule set is a collection of rules that share the same root entity and share common triggers. You write code for new rule set categories to define the triggers associated with a rule set. If you create rule sets for an existing rule set category (Event Messaging, Pre-update, and Validation), you do not need write trigger code.

Execution of rules always occurs at the rule set level. Unless the rules engine encounters an exit() command, all rules in the rule set are executed.

TrainingApp is built by heavily customizing an instance of ContactManager, and therefore has the same rule sets as ContactManager. These include:

- The Event Fired rule set, which is associated with MessageContext and triggers when a MessageContext object

Hierarchy: Rule Sets



- A **Rule Set** combines many individual rules into a useful set to consider as a group
- A Rule Set defines the root entity type
  - All the Rules in the set will be attached to same the triggering entity (root entity)
- Only the Rule Set Editor exposes individual rules.

Education

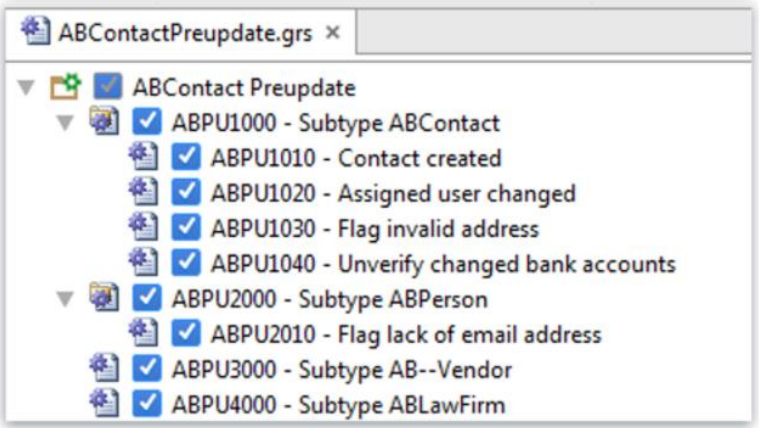
Notes

Hierarchy: Rules

If a rule has child rules, but the parent rule condition is false, neither the parent action nor the child rules are executed. In the slide example, the active (checked check box) rules are executed in the following hierarchy:

- ABPU1000 gathers together the rules relevant for all contacts. The child rules are executed only if this condition is true.
  - ABPU1010 creates a new history event when a contact is created.
  - ABPU1020 executes the necessary actions when a contact's assigned user changes (such as creating a note to record the change).
  - ABPU1030 executes the necessary actions when a flagged contact is unflagged (such as creating a note to record who unflagged the contact).
- ABPU2000 checks to see if the contact is an ABPerson. The child rules are executed only if this condition is true.
  - ABPU2010 flags any person who does not have an email address.

Hierarchy: Rules



- A **Gosu Rule** consists of a root entity, an expression that resolves to true or false, and an action that is executed if the condition is true
- Rules can have child rules
- If parent condition is true, Guidewire executes action and then executes all child rules

Education

Notes

What is the significance of a rule set's root entity?

Menu

Answer

What is the significance of a rule set's root entity?

Every rule set is tied to an entity, known as the root entity. This determines which class of objects the rule set is tied to. It also determines the object available to the rules in the rule set when the rule set is triggered.

Education

PREV NEXT

Notes

Gosu enhancements

Menu

Gosu enhancements

Education

PREV NEXT

Notes

Gosu enhancements

Guidewire types that can be enhanced include entities, Java classes, permissions, and SOAP entities. A fundamental reason for enhancing an entity is to add behaviors to a type that would not be appropriate to do by creating a subtype.

Gosu enhancements

- Gosu enhancements allow you to augment classes and other types with additional concrete methods and properties
- The most valuable and useful uses of this feature are:
  - to add new properties and methods to Guidewire Entities. Entities are defined in XML files and source code is automatically generated, thus cannot be modified, and
  - to define additional utility methods on a Java class or interface that cannot be directly modified. For example, the source code is unavailable, or a given class is final (cannot be subclassed)
- If an Entity/Java class is enhanced, then all the subtypes/subclasses will automatically inherit all elements from the enhancement

Education

Notes

Enhancement components

Enhancements can have any number of the following: getters, setters, and methods.

Menu

Enhancement components

Getter properties	Setter properties	Functions
• Calculate derived values	• Take input parameters, and/or	• Set field values that require additional logic • Modify other objects, and/or • Create objects

Education

Notes

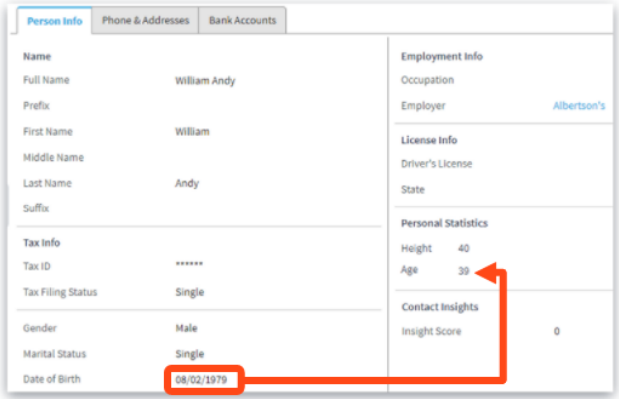
Derived (virtual) properties

In the slide example, ABPersonEnhancement.gsx is the enhancement file that extends the ABPerson entity with an Age property.

Menu

Derived (virtual) properties

- ABPersonEnhancement.gsx enhances ABPerson entity and defines the Age getter property
- Property calculates age based on the date of birth value
- Derived values should NOT be stored in the database, because storing them would be redundant
  - They are calculated from physical fields



Education

Page 6

PREV NEXT

Notes

Getter property

A getter is appropriate when you can derive a value and the derived value does not need to be stored in the database. In the slide example, ABPersonEnhancement.gsx is the enhancement file that extends the ABPerson entity with an Age property.

Menu

Getter property

- A **getter property** is used to calculate a derived value
  - Property not declared at Data Model level
    - As a result the value is not stored in database
  - Code cannot receive input parameters
  - Code can only return a value and should not alter any data
  - Null safe
- Example: `ABPerson.Age`
  - Derives value by calculating number of years between date of birth and current date
  - Returns "Unknown" if date of birth is null
  - Storing Age in database is redundant

Education

Notes

Setter property

An implementation of any Guidewire application will have a small number of setters because of the limiting criteria for use. A setter is appropriate only when you need to set one or more fields based on a single input value and—if there is a single field to be set—it is a field that cannot be set by simply assigning the input value to the field.

Possible use cases for setters could include:

- Adding a value to one field out of several possible fields (such as adding a new phone number to either Work Phone, Home Phone, or Cell Phone).
- Adding an object to an array and making that value the primary object (such as adding a new employee to an ABCompany and making that employee the primary contact).
- Adding a value to a field that must be converted to a different unit (such as receiving a dollar amount in Euros but needing to store that amount as dollars).
- Converting an integer value to a typekey value (such as receiving an integer that represents the number of employees and using that to set a CompanySize typekey value which could be: Under 100, 101 to 500,

Setter property

- A **setter property** takes a single input value and uses it to modify the associated object
  - Property not declared at data model level
  - Value given to setter may or may not be stored in database
  - Value manipulated by setter is typically stored in database
  - Code must receive exactly one input parameter
  - Should not be used to alter other objects
- Example: `ABPolicyPerson.HeightInInches(arg)`
  - Getter takes height from database (in meters) and converts it to inches (so, for example, it can be displayed in UI)
  - Setter takes height (in inches) and converts it to meters before saving it to database

Education

Notes

Gosu enhancements for Java

Gosu enhancements provide additional methods (functionality) on a Guidewire entity. For example, suppose that you create an enhancement to the Activity entity. Within this enhancement, you add methods that support new functionality. Then, if you type Activity. (Activity dot) within any Gosu code, Studio automatically uses code completion on the Activity entity. It also automatically displays any methods that you have defined in your Activity enhancement, along with the native Activity entity methods.

Menu

Gosu enhancements for Java

- Write Gosu enhancements for Java types
- Useful when creating generic methods that…
  - Do NOT relate to a specific entity
  - Extend the functionality of a base data type
- Examples:
  - GWBaseIntegerEnhancement enhances `java.lang.Integer`
  - GWBaseDateEnhancement enhances `java.util.Date`
  - GWBaseListEnhancement enhances `java.util.List`
- Java enhancements typically contain static functions
- Deploy and debug the same way as other enhancements

Education

Notes

What type of logic does a getter implement? A setter? A method?

Menu

Answer

What type of logic does a getter implement? A setter? A method?

The correct responses are:

Getter: Logic that returns a derived value and does not take parameters or change other data.

Setter: Logic that takes a single input value and uses that to modify some other field or set of fields on the given object.

Method: Logic that requires parameters, changes data, or otherwise does more than simply deriving or setting a value.

Education

PREV NEXT

Notes

About Pattern Matching

This section discusses pattern matching.

Menu

About Pattern Matching

Education

Notes

Pattern matching

Validation is a general application behavior that prevents a user from saving invalid business data. Field-level validation is a validation behavior tied to one or more specific data fields.

A regular expression defines a pattern. The save is not allowed if the data entered by the user doesn't match the pattern.

Input masks both display a watermark in the field and guide the user to input the correct format of data. An input mask does not restrict data commit, but it does guide the user to input the data in a certain format.

Pattern matching

- Pattern matching is a field-level validation technique
- Verifies that the user entered the data in the correct format for that field
  - E.g.: tax id must be a nine-digit number with hyphens after the third and fifth digits
  - Prevents the user from saving invalid business data
- Could be implemented in the data model or the UI tier
- Main components:
  - Regular expression
  - Input mask (optional)

Notes

Pattern matching - regular expressions

The pattern for the Email validator is:

- Any non-empty string, specified as: .+
- An "at" symbol, specified as: @
- Any non-empty string, specified as: .+
- A period, specified as: \.
- Any non-empty string, specified as: .+

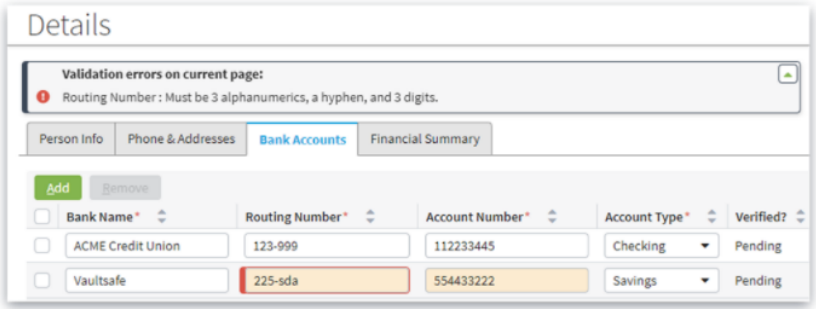
Characters that are not treated as literals and must be escaped include: [ ] ( ) { } . * + ?

For a complete listing of syntax for validator patterns, consult the Configuration Guide for your product.

Menu

Pattern matching - regular expressions

- A regular expression defines an abstract pattern
- Examples:
  - Email address: `.+@.+\..+`
  - Routing number: `[0-9a-zA-Z]{3}-[0-9]{3}`



Education

PREV NEXT

GUIDEWIRE

Notes

Pattern matching – input masks

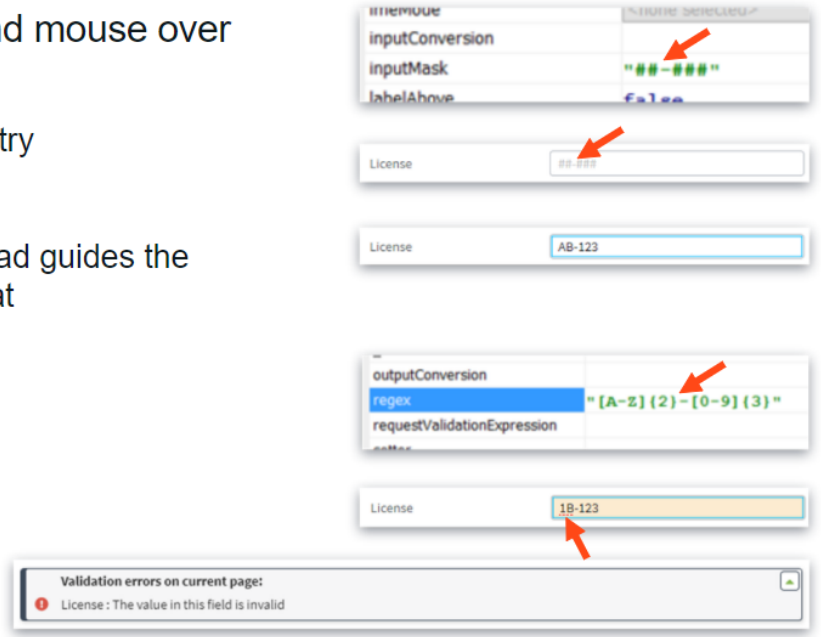
Input masks and regular expressions guide the user input for the field. Client-side regular expressions notify the user of an issue with the field input when a placeholder character (#) is used in the inputMask expression. An input mask identifies to the user the specific form of the data to enter into a given field. Typically, an input mask shows a basic pattern for a field. Input masks can also guide the user to input a specific character input in the field.

When an input mask contains placeholders (e.g., #) but does not specify either a regex property or does not have an associated entity field validator, the framework generates an implicit regular expression based on the inputMask property definition. As the user types in the field, the implicit regex triggers client-side validation, but does not restrict data commit. Instead, it guides the user to input the data in a certain format. In the example above, the field value should be 5 characters in length, the first two being upper case letters and the last three being numbers.

It should be noted that in addition to placeholders (#) in an inputMask property expression, you can force certain characters to be in the data. For example, "AB-###" forces the field to have the upper-case letters A and B to be the first two characters in the entry.

Pattern matching – input masks

- Appears as a watermark in field and mouse over tooltip
  - Changes to a darker color with data entry
  - No format warning
  - Does not restrict data commit but instead guides the user to input the data in a certain format



Education

Notes

Pattern matching- practices comparison

Shown here are the pros and cons and pattern matching in the Data Model and the User Interface.

Menu

Pattern matching- practices comparison

Data model

- Pros
  - Application enforces compliance in the UI and through API
  - Supports full localization
  - Robust extensibility
- Cons
  - Static values only: regex and input mask cannot change based on business logic
- Implemented as a reusable field validator

User interface

- Pros
  - Dynamic values: regex and input mask can change based on business logic
  - Can restrict data commit
- Cons
  - Only a few widget support it
  - Must be configured per widget
  - No custom error message option
  - Extensibility a big concern
- Implemented as widget properties

Education

PREV NEXT

Notes

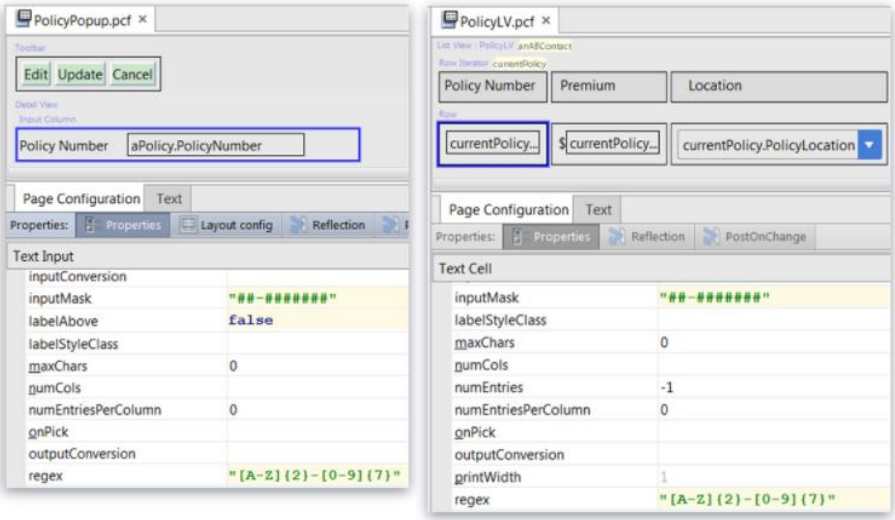
Example: pattern matching in the UI

PCF Format Reference: <application-installation-folder>/modules/pcf.html

Menu

Example: pattern matching in the UI

- Implementing the user story in the UI requires duplicating regex and input mask
- **inputMask** and **regex** properties can be dynamic
  - To improve the design, a function call could be used instead of hard-coding the strings
- No custom error message
- Not all the widgets have the **regex** and **inputMask** properties
  - Refer to PCF Format Reference

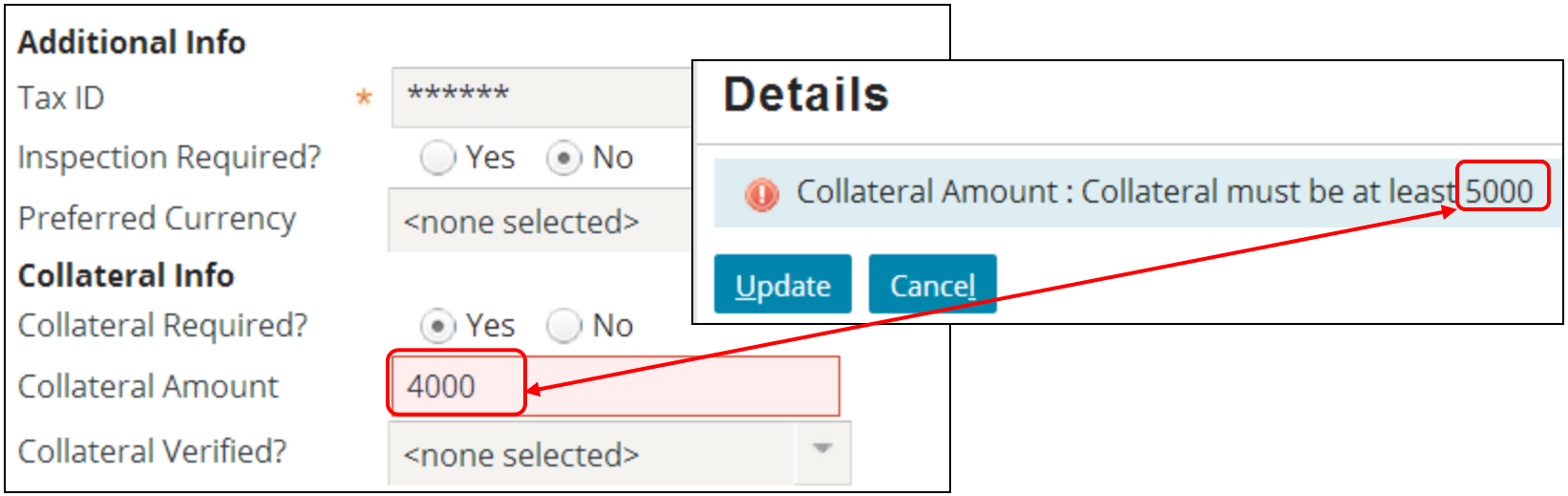


Education

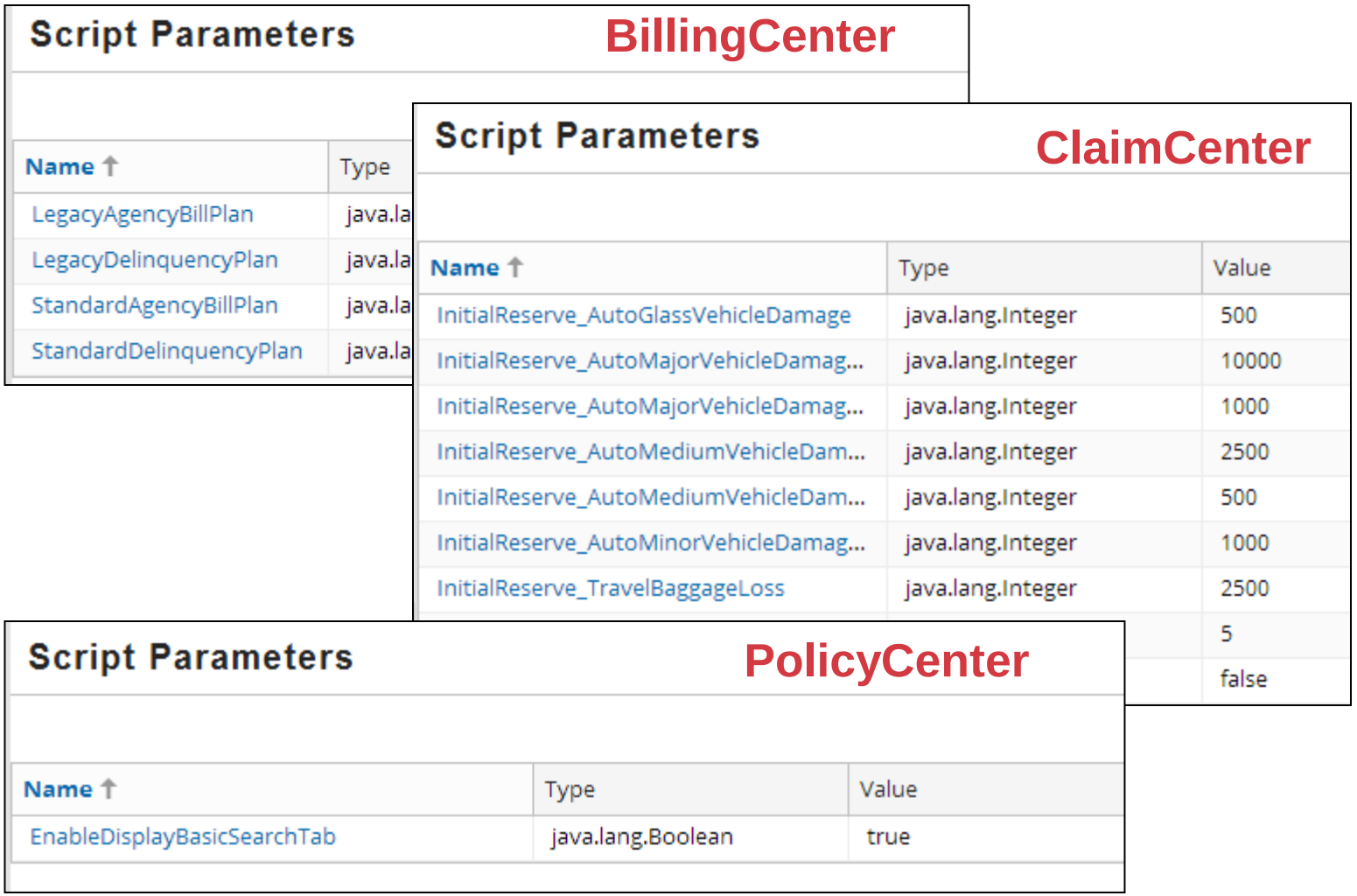
PREV NEXT

Script parameters

- A **script parameter** is an application-wide global constant
  - Value maintained by administrator
  - Can be referenced in any Gosu code
  - Typically used for values that may change over time and therefore should not be hard-coded



Examples from base applications



Example 2: Toggling application behavior

```
// This functions creates a history event identifying that the given
// contact was viewed by the current user. (This function does nothing
// if the RecordInHistory-UserViewsOfContacts script parameter is set to false.)

   if (ScriptParameters.RecordInHistory_UserViewsOfContacts) {
```

when set to true...

History

Date	Event Type	Description
09/19/2013	Viewed	Alice Applegate viewed this contact.
09/19/2013	Viewed	
09/18/2013	Viewed	
09/18/2013	Viewed	
09/16/2013	Viewed	
09/16/2013	Created	

when set to false...

Date	Event Type	Description
09/18/2013	Viewed	Super User viewed this contact.
09/16/2013	Viewed	Super User viewed this contact.
09/16/2013	Viewed	Super User viewed this contact.
09/16/2013	Viewed	Super User viewed this contact.
09/16/2013	Created	Contact William Andy created

Notes

About Partial Page Update

This section discusses partial page update.

Menu

About Partial Page Update

Education

PREV NEXT

Notes

Partial page update: Layout re-render

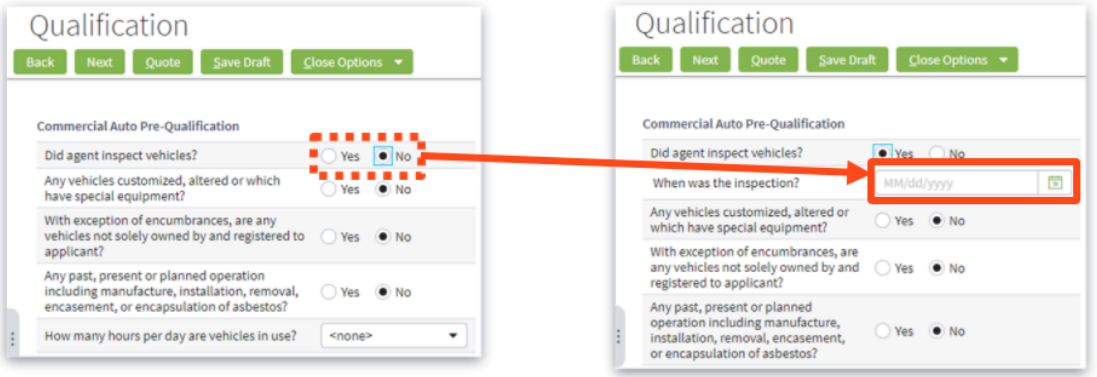
In the slide example, the Company Info card has both an Inspection Required field and Inspection Date field. However, the Inspection Date field is visible only if the Inspection Required? field is set to Yes.

The InspectionRequired widget displays a Yes or No radial buttons. A selected Yes radial button equates to true and a selected No radial button equates to false. A null value is represented as neither radial button selected.

Dynamic widget behavior differs from a property that evaluates an expression in that the user changes the business data while editing it and that change affects the behavior of the widget. In the previous examples, the changes are present only after the data has been committed.

Partial page update: Layout re-render

- Responds to a user changing business data **while** it happens and changes the layout of widgets on the screen
- Example:
  - While a user changes the Did agent inspect vehicles? field to Yes, a secondary question and the Date Input widget becomes visible



Education

PREV NEXT

Notes

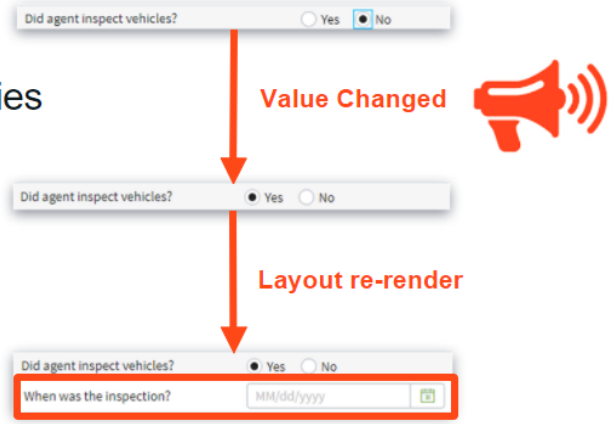
Partial page update: Layout re-render (Continued)

There are two general and non-exclusive categories for a partial page update: layout re-render and DATA_ONLY.

In the layout re-render example above, whenever the value of the InspectionRequired field is changed, all user editable data is sent from the client to the application server. No data is committed. The application server processes the request and returns instructions and user editable data to the client. The client re-renders the layout that shows an Inspection Date field.

Partial page update: Layout re-render (Continued)

- Layout re-rendered for the page
- No data committed
- Applies to following dynamic widget properties
  - visible
  - editable
  - available
  - required



Education

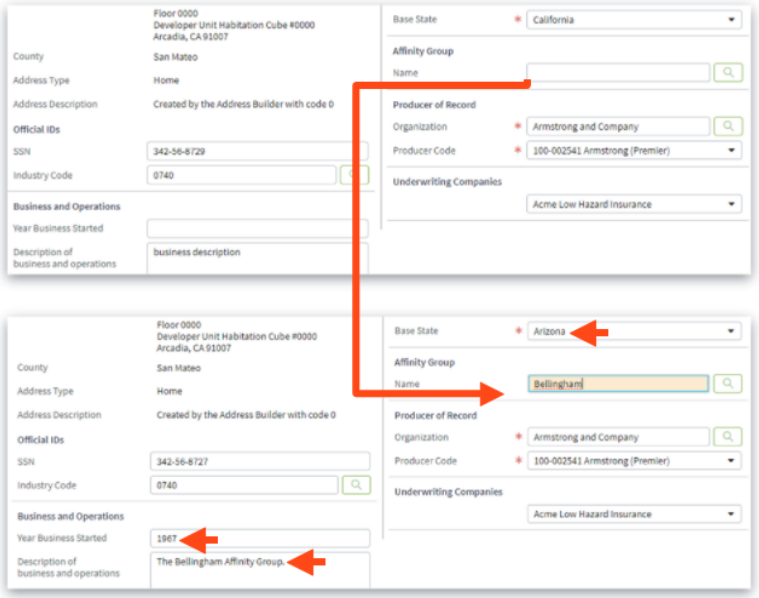
Notes

Partial page update: DATA_ONLY

In the DATA_ONLY example, the widget configuration enables dynamic widget behavior for the Affinity Group name TextInput widget. When a user changes the value of the Name field, the client makes a request to the application server. The application server processes the request and returns instructions to the client. In this case, the instructions tell the client to render only the newly supplied data for the Business and Operations fields (year started and description of business)

Partial page update: DATA_ONLY

- Responds to a user changing business data **while** it happens
  - Not after the user navigates to the page
  - Not after the user commits or tries to commit the data
- Example:
  - While a user changes the Affinity Group Name field, values for other fields are rendered with changes (read-only in UI)



Education

Notes

Two kinds of widget properties

In the slide example, the id property of a widget is a static property. The widget always has the same ID, regardless of the state of the application or the values of any business data. The property does not evaluate an expression, just the value of a type, such as a String or Integer value.

The editable property evaluates a boolean expression and the expression can be a dynamic value, in this case, one that returns true or false. The label property evaluates a string expression. This lesson discusses display keys later on. Display keys are a type of string expression in Guidewire applications for localizing text. The value property is also a dynamic property. The value property evaluates an object expression that binds the object property to it.

Both types of widget properties can be required properties for a specific type of widget.

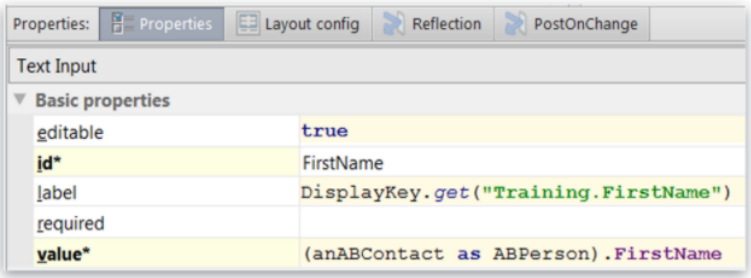
Two kinds of widget properties

Static Property

- Evaluates a static value that is immutable, never changes
  - id requires a static value

Dynamic Property

- After a user navigates to a page or clicks update, evaluates an expression and returns a value
  - editable requires boolean expression
  - label returns a string expression
  - value returns an object expression



Education

Notes

Dynamic properties evaluate expressions

Not all widgets have all properties. Refer to the PCF Format Reference for more examples. Recall that the PCF Format Reference defines the properties for every widget, including the type of value the property takes. Note that the PCF Format Reference refers to widget properties as "attributes". To open the PCF Format Reference, open <ApplicationRootDirectory>\modules\pcf.htm file in your web browser.

You can also consider the value property of a widget to be a dynamic property because it evaluates an object expression.

Visible is typically not set to false. It is either set to true or it is set to an expression that renders the widget visible or not visible. The default value for the visible property is true.

Dynamic properties evaluate expressions

- Available
  - Boolean expression which, if false, grays out the widget and its children
- Editable
  - Boolean expression which, if false, makes the widget and its children read-only
- Required
  - Boolean expression which, if true, then a value must be filled out by the user
- Visible
  - Boolean expression which, if false, completely hides the widget and its children

Notes

Post On Change properties

In certain cases, you may want to toggle (disable or enable) targeted Post On Change behavior for a specific widget.

In the slide example, the Invoicing Method field will disable Post On Change if the currency is not USD.

For all other currencies, the Post On Change will be disabled. The widget will not announce a change in its value; data will not make a round-trip between client and the application server; the page will not be re-rendered and data won't be refreshed.

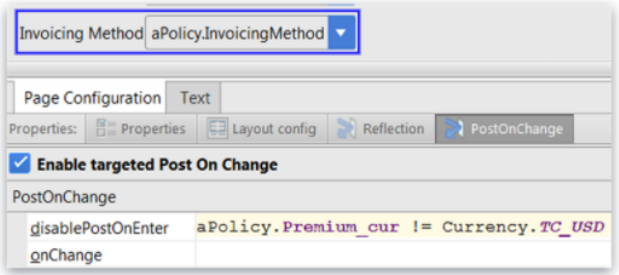
In the slide example, the Gosu expression defined for the onChange property invokes the setFieldsOnResolution() function. The function is defined in the entity enhancement, FlagEntryEnhancement.gsx.

```
16 /* This function is called when
a FlagEntry's resolution field
17 is set. This function sets the
UnflagDate and UnflagUser
18 fields. This serves the role of a
"FlagEntry Pre-Update"
19 rule set.
20 */
21 function
setFieldsOnResolution() : void {
22 this.UnflagDate =
gw.api.util.DateUtil.currentDate()
23 this.UnflagUser =
User.util.getCurrentUser()
24 } // end of function
```

Post On Change properties

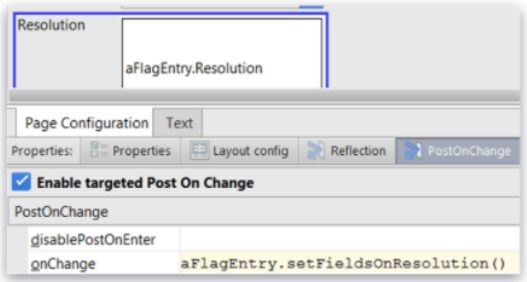
disablePostOnEnter

- If evaluated true when page is rendered, this field won't trigger Post On Change
- Default is false



onChange

- Defines a Gosu expression to invoke when user changes the value of the widget
- Causes immediate post back to the server



Education

Notes

Target property (earlier versions)

When specified as a widget, the target area can be a specific widget or a grouping of widgets. For example, if the Widget ID corresponds to an Input Set, then the widgets in the input set are re-rendered.

There is an additional layout re-render configuration for backwards compatibility: no value. The no value configuration consists of enabling targeted post on change alone with no property configurations. The backwards compatibility configuration has no property values specified and causes the re-rendering of the entire page and refreshes all user-editable data. There is a high-performance cost for implementing targeted Post On Change without specifying a target and any other properties.

In current InsuranceSuite applications, targeted update is automatic for any widget where Targeted Post On Change is enabled, and the target property is redundant.

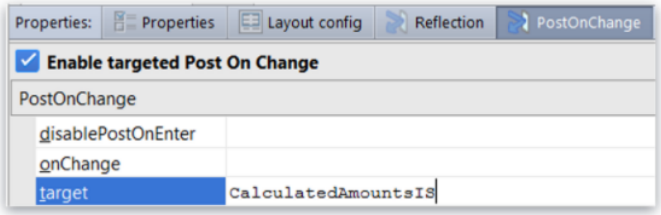
Target property (earlier versions)

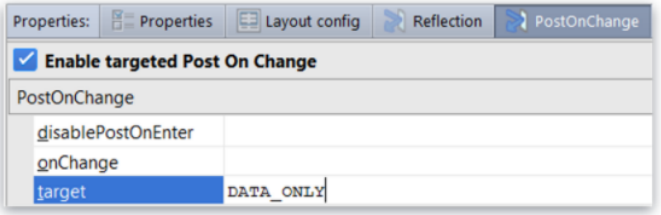
Layout re-render

- Widget ID
- Target area re-rendered
- All editable data refreshed
- Widgets can also be grouped and parent container can be targeted

Data update

- DATA_ONLY
- No layout re-render
- All editable data refreshed
- Better performance





Education

Notes

Name two properties that evaluate expressions.

Menu

Answer

Name two properties that evaluate expressions.

Possible answers include:

- editable
- required
- visible
- available

Education

PREV NEXT

Notes

How is a partial page update implemented?

Menu

Answer

How is a partial page update implemented?

A partial page update is implemented by configuring dynamic widget behavior by enabling targeted post on change. Often the dynamic widget behavior configuration also includes configuring an expression for a widget property.

Education

PREV NEXT

Notes

What are the three properties that you can configure for a widget with targeted Post On Change enabled?

Menu

Answer

What are the three properties that you can configure for a widget with targeted Post On Change enabled?

disablePostOnEnter, onChange, and (with earlier InsuranceSuite versions) target.

Education