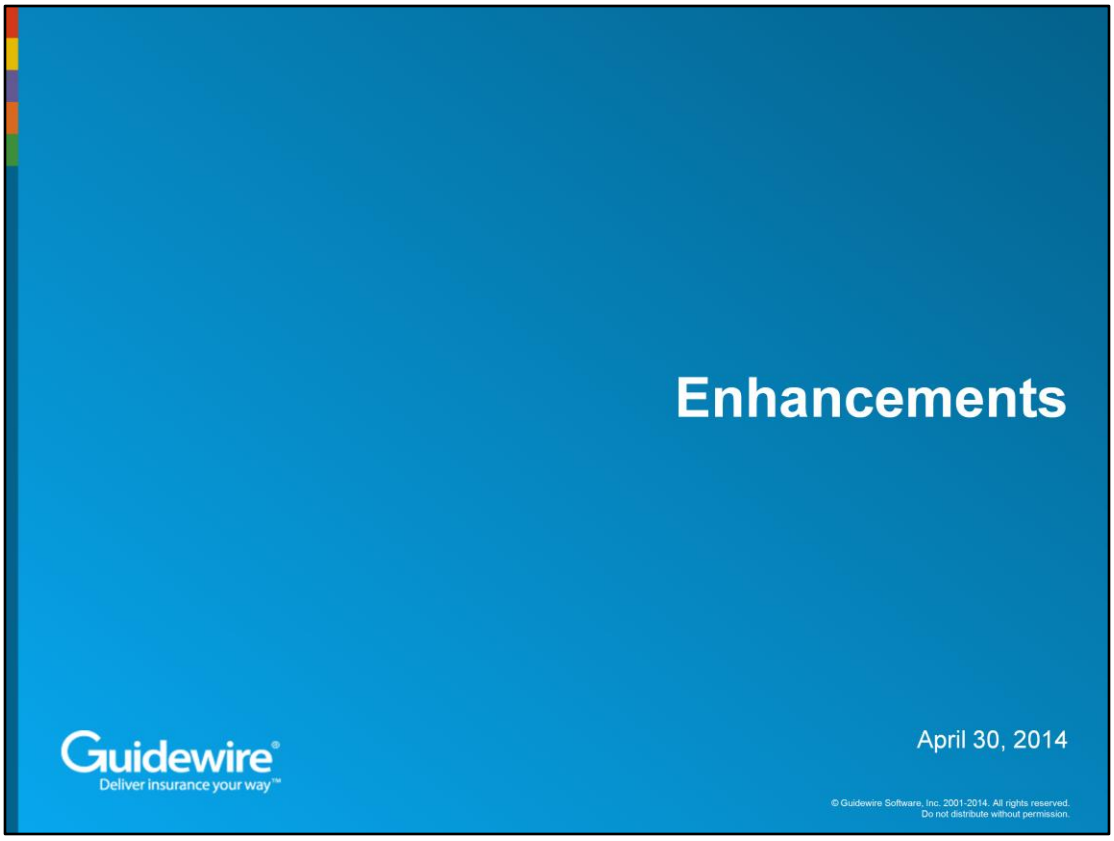


Guidewire training materials contain Guidewire proprietary information that is subject to confidentiality and non-disclosure agreements. You agree to use the information in this manual solely for the purpose of training to implement Guidewire software solutions. You also agree not to disclose the information in this manual to third parties or copy this manual without prior written consent from Guidewire. Guidewire training may be given only by Guidewire employees or certified Guidewire partners under the appropriate agreement with Guidewire.

Lesson objectives

- By the end of this lesson, you should be able to:
  - Describe the purpose and functionality of Gosu enhancements
  - Create entity enhancements
  - Reference entity enhancement properties and methods
  - Create Gosu enhancements for Java classes and interaces
  - Debug enhancements

This lesson uses the notes section for additional explanation and information.
To view the notes in PowerPoint, select View → Normal or View → Notes Page.
When printing notes, select Note Pages and Print hidden slides.

Lesson outline

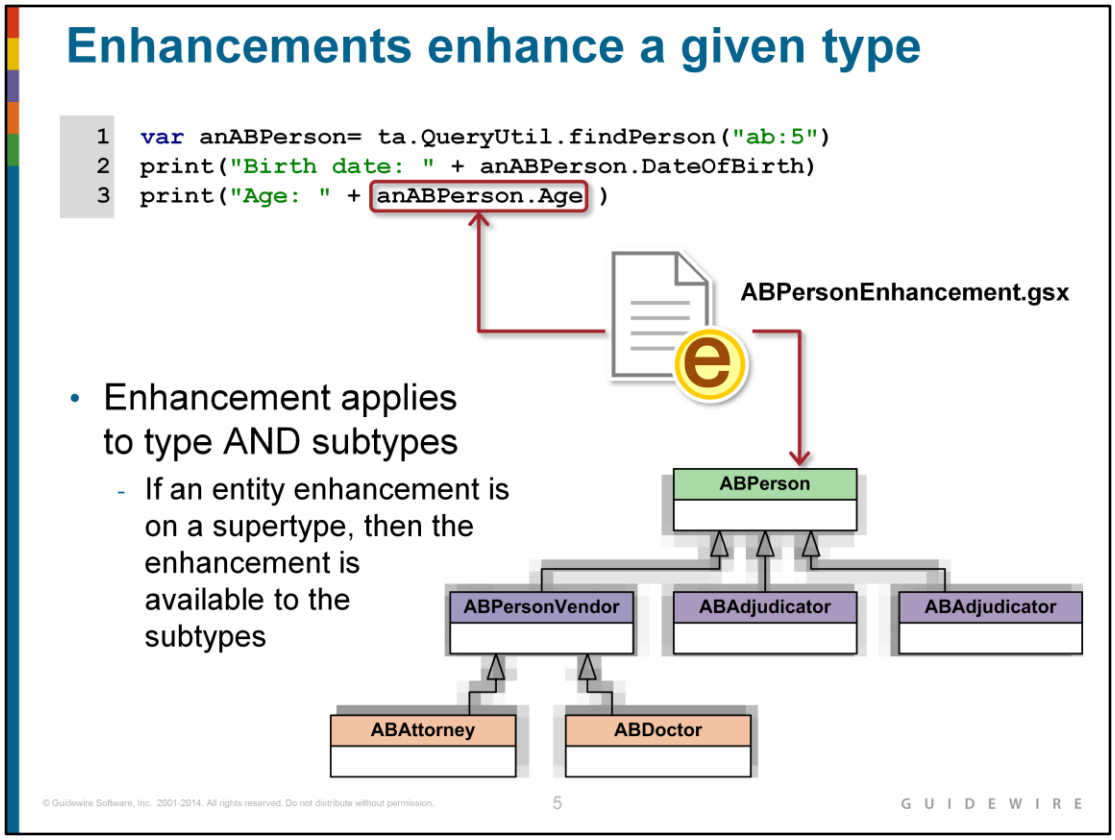
- Entity enhancement fundamentals
- Working with entity enhancements
- Debugging enhancements
- Gosu enhancements for Java

Gosu Enhancements

- A **Gosu enhancement** is a set of code that enhances the functionality of an existing Guidewire type
- For configuration developers, nearly all enhancement work involves enhancing these types
- Lesson focuses on enhancements for Guidewire entities and Java classes

Guidewire types that can be enhanced include entities, classes, PCF files, permissions, SOAP entities, and typekeys.

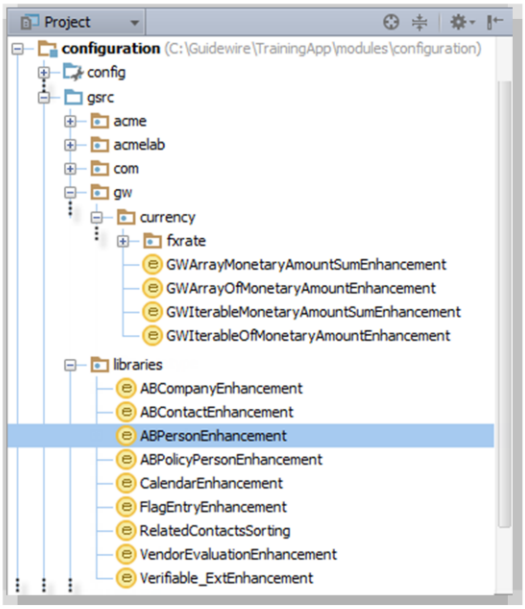
Gosu enhancements can also be used to enhance types where sub-classing is impractical or not possible.



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

The property is available to any ABPerson or any object that is a subtype of ABPerson such as ABDoctor. However, the Age property it is not available to any other entity such as ABCompany.

Enhancement files



- Guidewire and customer code
  - …\configuration\gsrc\
- Organized into packages
  - Classes
  - Enhancements

- .gsx files

A package is an aspect of object oriented programming languages. It is a collection of classes gathered together for convenience often because the classes have a similar purpose. Because enhancements extend the functionality of an entity class, enhancements are also stored in packages.

A package can contain any number of enhancements and/or Gosu classes. In most Guidewire applications you can find enhancements in the libraries folder and the gw sub-packages.

Enhancement components

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

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

Enhancements to get values

- Derived values should NOT be stored in the database

- ABPersonEnhancement.gsx defines the Age property
- Property calculates age based on the date of birth value

In the slide example, ABPersonEnhancement.gsx defines the Age property.

Getter properties

- A **getter property** is used to calculate a derived value
  - Property not declared at data model level
  - Value not stored in database
  - Code cannot receive input parameters
  - Code can only returns a value
  - Should not be used to alter any data
  - Null safe
- Example: **`ABPerson.Age`**
  - Derive 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

A getter is appropriate when you can derive a value and the derived value does not need to be stored in the database.

Example getter: ABPerson.Age

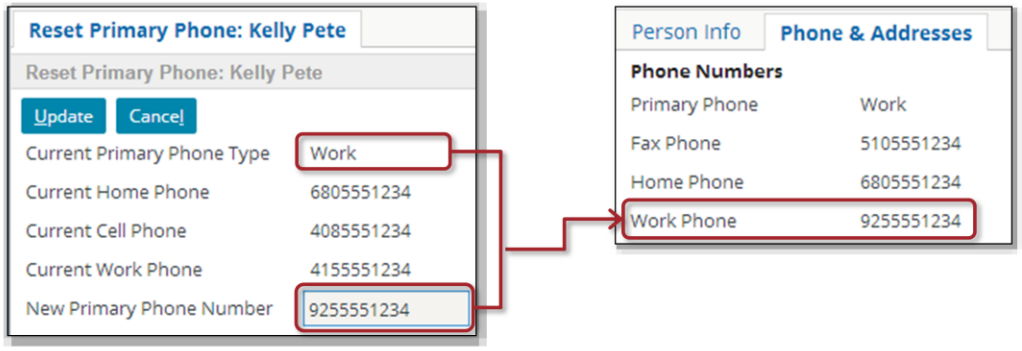
```
27  property get Age(): String {
28    if (this.DateOfBirth == null) {
29      return "Unknown"
30    } else {
31      var today = DateUtil.currentDate()
32      var ageInDays = DateUtil.daysBetween(this.DateOfBirth, today)
33      var ageInYears = Math.roundDown(ageInDays / 365)
34      return ageInYears as java.lang.String
35    }
36  }
```

- Enhancement declaration identifies type it enhances
- Keyword **`this`** represents the object from which to call the property

For all enhancement logic, the this keyword represents the object from which the getter, setter, or method is called.

In the slide example, ABPersonEnhancement.gsx defines the Age getter property.

Enhancements to set values



- Use setters to modify values for calling object
- Example:
  - Setter determines which phone field to update using the primary phone typekey for an ABPerson object

In the slide example, work is the primary phone typekey for the contact. The new number will update the work phone field rather than the cell, fax, or home phone fields. ResetPrimaryPhoneWorksheet.pcf contains the setter reference.

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
  - Values manipulated by setter are 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

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, 501 to 1000, Over 1000).

Example setter

```
44  property set NewPrimaryPhone(newPhoneNumber: String) {
45    if (this.PrimaryPhone == typekey.PrimaryPhoneType.TC_HOME) {
46      this.HomePhone = newPhoneNumber
47    }
48    if (this.PrimaryPhone == typekey.PrimaryPhoneType.TC_WORK) {
49      this.WorkPhone = newPhoneNumber
50    }
51    if (this.PrimaryPhone == typekey.PrimaryPhoneType.TC_MOBILE) {
52      this.CellPhone = newPhoneNumber
53    }
54  }
```

- Must receive exactly one value, which is value to set
- Keyword **`this`** represents the object from which to call the property

For all enhancement logic, the this keyword represents the object from which the getter, setter, or method is called.

In the slide example, ABPersonEnhancement.gsx defines the NewPrimaryPhone property.

Enhancement methods

- A **method** is a set of statements that executes a logical unit of work
  - Can receive any number of parameters
  - Can optionally return a value
- Create when your code…
  - Requires multiple input parameters, and/or
  - Modifies multiple unrelated fields on given object, and/or
  - Creates or modifies other objects, and/or
  - Does not "feel like" a setter
- Example: **`ABPerson.assignDefaultOccupation()`**
  - Uses information known about object to generate default occupation

There are two elements that alter data: setters and methods. Each is appropriate under different circumstances.

A setter is appropriate when you are making a change to the object, the change "feels like" you are setting a field, a single input value is used to determine how to set the value, and no other objects get changed. An example of a setter is Contact.PrimaryAddress(anAddress). The setter adds anAddress to the contact's array of addresses (if necessary) and then sets that address to be the primary address.

A method is appropriate when you need to make a change and either the change doesn't feel like you are setting a field, you need multiple parameters, and/or there are changes that include other objects An example of a method is Contact.CreateFraudCase(Reason). The method creates an activity associated to the contact and creates a note associated to the activity that includes the Reason in the note body.

Example method (1)

```
64  function assignDefaultOccupation(): void {
65    if (this.Occupation == null) {
66      if (this.Subtype == typekey.ABContact.TC_ABATTORNEY) {
67        this.Occupation = "Attorney"
68      } else {
69        if (this.Subtype == typekey.ABContact.TC_ABDOCTOR) {
70          this.Occupation = "Doctor"
71        } else {
72          if (this.Employer != null) {
73            this.Occupation = "Employee of " + this.Employer.Name
74          } else {
75            this.Occupation = "Unknown"
76          }
77        }
78      }
79    }
80    // end null check
81  }
```

- Keyword **`this`** represents the object from which to call the property

For all enhancement logic, the this keyword represents the object from which the getter, setter, or method is called.

In the slide example, ABPersonEnhancement.gsx defines the assignDefaultOccupation() enhancement method.

Example method (2)

```
88  function addContactNote(): ContactNote {
89    var newContactNote = new ContactNote()
90    this.addToContactNotes(newContactNote)
91    return newContactNote
92  }
```

- Create a new instance of an entity (object) in method and relate the new object to an existing object
  - Set one object's foreign key to the other object (harder)
  - Add the new object to an existing array (easy)
- Example:
  - Line 89: Creates new object
  - Line 90: Adds new object to the object that called the method

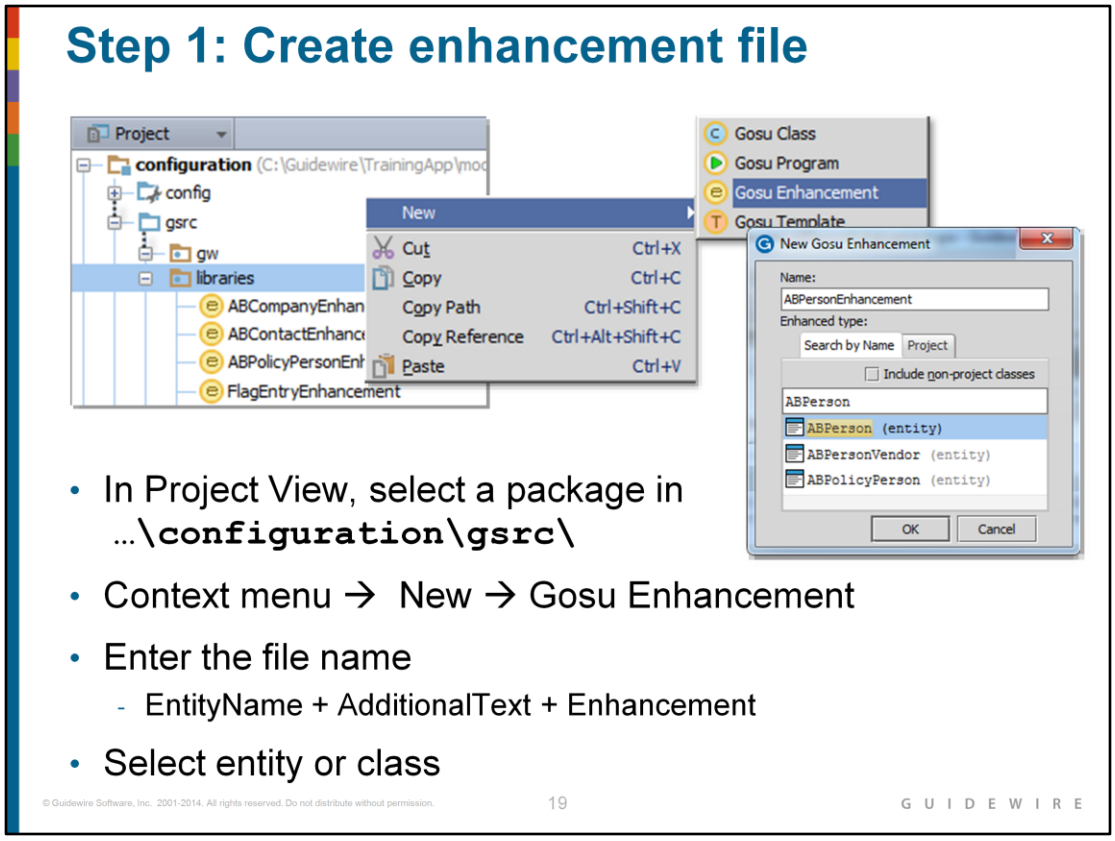
In the slide example, ABContactEnhancement.gsx defines the addContactNote() enhancement method.

Lesson outline

- Entity enhancement fundamentals
- Working with entity enhancements
- Debugging enhancements
- Gosu enhancements for Java

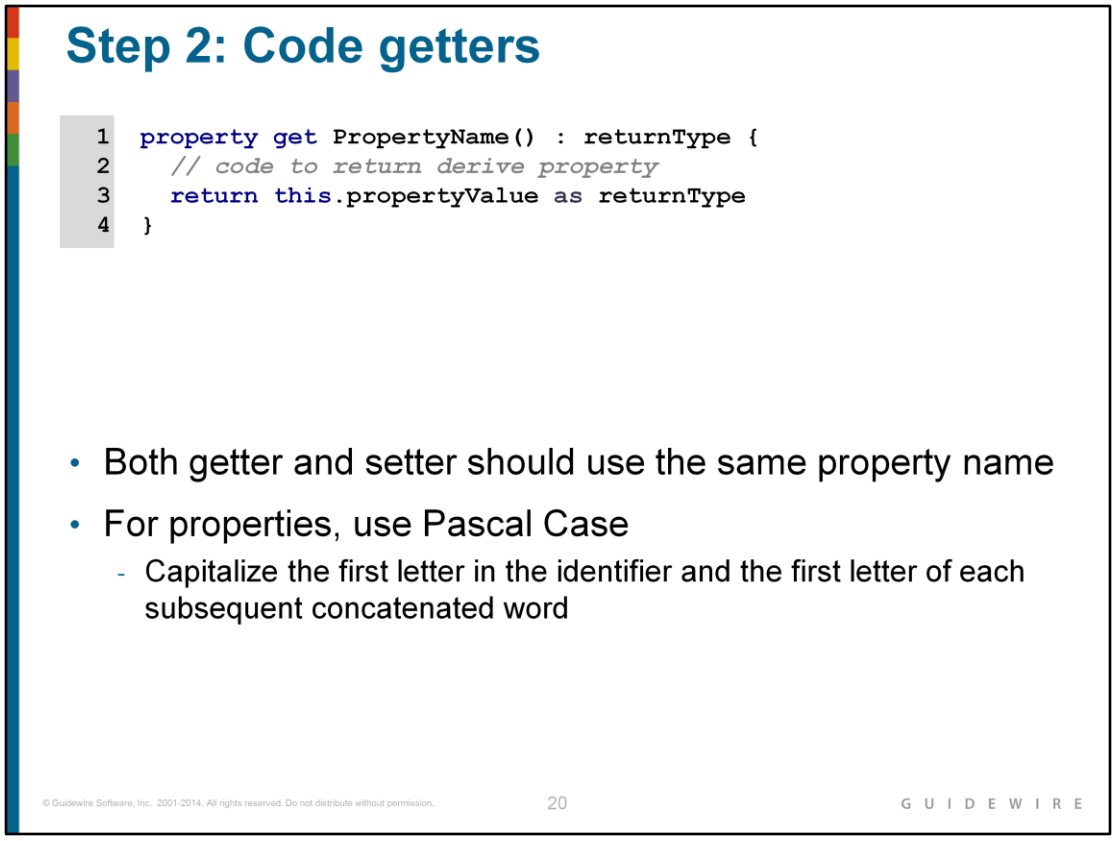
Steps to implement an enhancement

1. Create a new enhancement file
2. Code getter properties, setter properties, methods
3. Deploy the enhancement
4. Reference the enhancement



Guidewire recommends that you create an enhancement in a custom package named after your company. For example, if you are implementing an application for Acme Insurance, then you would create all custom enhancements in a package named Acme or in sub-packages of an Acme package.

The recommended naming convention for enhancements is <EntityName> + <AdditionalTextWhenNeeded> + Enhancement. In some cases, you may need to have multiple enhancements for a single entity. In this case, the <AdditionalTextWhenNeeded> should be used to clarify the difference between the enhancements.



The recommended capitalization convention for properties is to use Pascal Case. In Pascal Case, the first letter in the identifier and the first letter of each subsequent concatenated word are capitalized.

You can precede the property keyword with one or more modifier keywords. Guidewire modifiers include public and private, which are access modifiers. A property is public by default, meaning that it can be referenced from anywhere in a Guidewire application that uses Gosu.

In contrast, a private property can be referenced only within the library in which it is defined. Because the public keyword is optional, it has been omitted in the slide example.

For more information about modifiers, see the Gosu Reference Guide.

Step 2: Code setters

```
1  property set PropertyName(parameter : dataType) {
2    // code to set property with a value
3    var newValue  = parameter + parameter
4    this.propertyValue = newValue
5  }
```

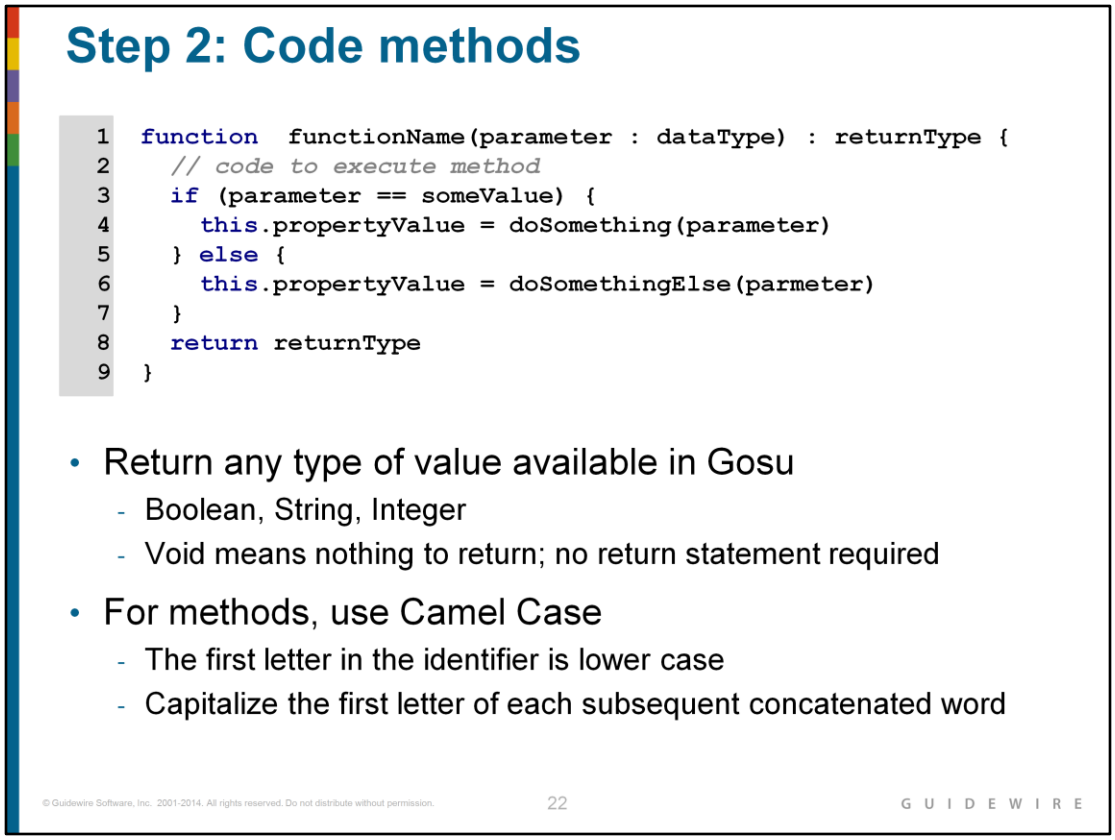
- Both getter and setter should use the same property name
- For properties, use Pascal Case
  - Capitalize the first letter in the identifier and the first letter of each subsequent concatenated word

The recommended capitalization convention for properties is to use Pascal Case. In Pascal Case, the first letter in the identifier and the first letter of each subsequent concatenated word are capitalized.

You can precede the property keyword with one or more modifier keywords. Guidewire modifiers include public and private, which are access modifiers. A property is public by default, meaning that it can be referenced from anywhere in a Guidewire application that uses Gosu.

In contrast, a private property can be referenced only within the library in which it is defined. Because the public keyword is optional, it has been omitted in the slide example.

For more information about modifiers, see the Gosu Reference Guide.



The recommended capitalization convention for methods is to use Camel Case. In Camel Case, the first letter of an identifier is lowercase and the first letter of each subsequent concatenated word is capitalized. Examples include popupButtonText, calculateAvailability, or assignToNextAvailableUser.

A method can return any type of value available in Gosu, including Boolean, String, and Integer.

Like the property keyword, you can precede the function keyword with one or more modifier keywords. Guidewire modifiers include public and private, which are access modifiers. A method is public by default, meaning that it can be referenced from anywhere in a Guidewire application that uses Gosu.

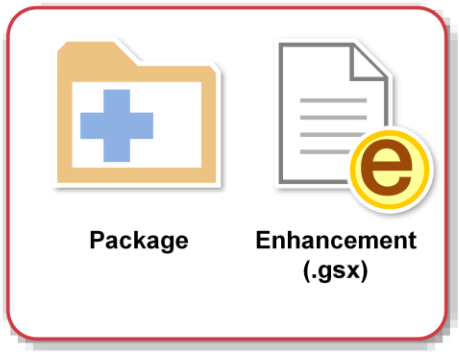
In object-oriented languages such as Gosu, you can create overload a method in the same class. Overloading means creating multiple methods with the same name and the same return type, but each method signature has a different number of input parameters and/or different data types.

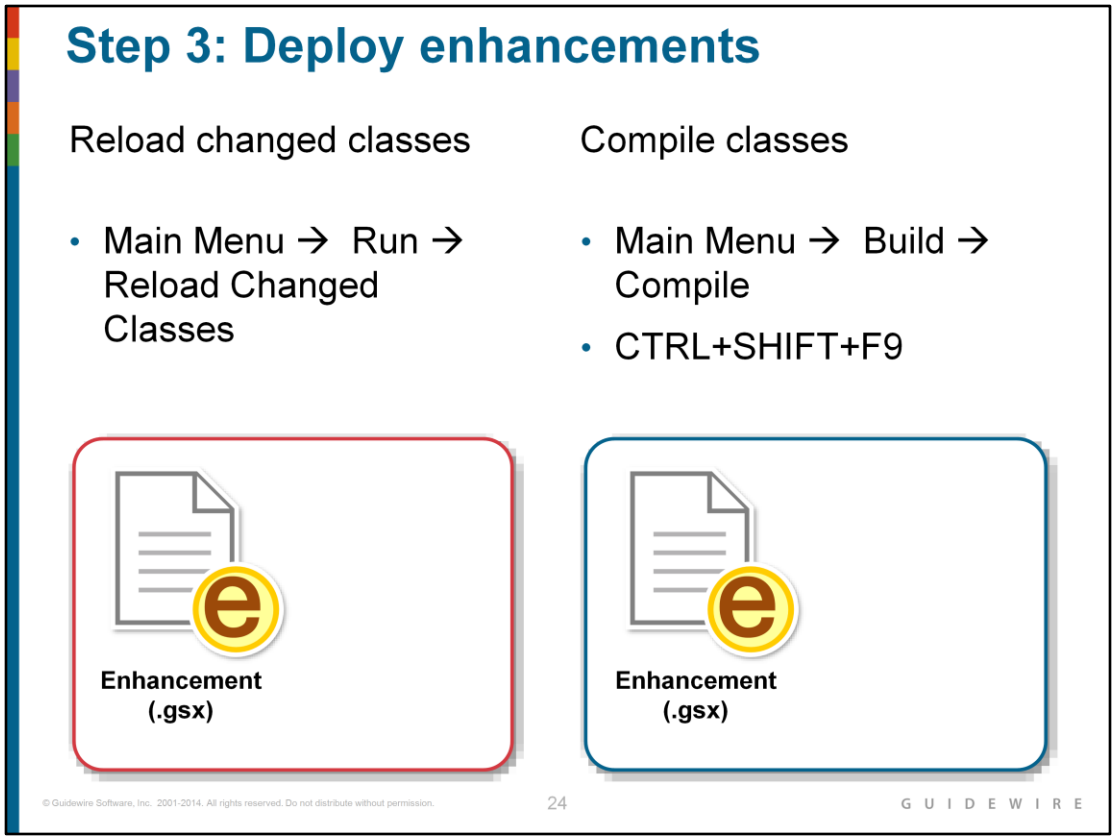
If editing a base entity enhancement, only to create new methods. Do not to overload existing methods that exist already within the application.

Step 3: Deploy package and enhancement *New*

Restart Server

- New package and enhancement loaded at server startup

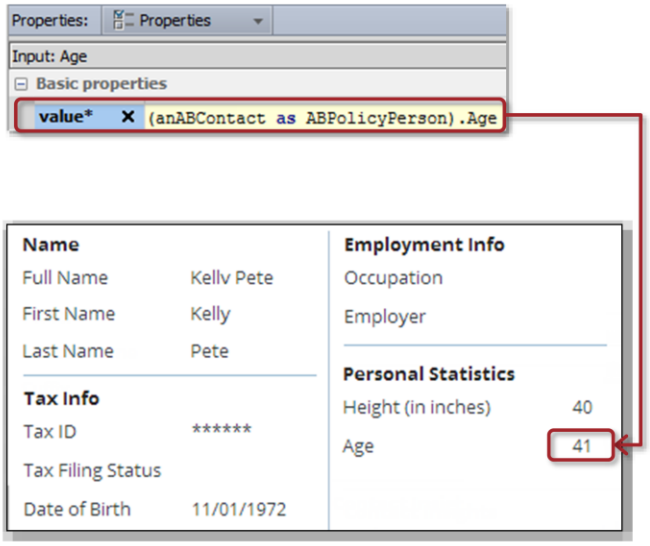




You can deploy new and modified enhancements when the server is running in run or debug server process. If the enhancement exists in a new package, then you must restart the server.

If your modified enhancement (.gsx) contains a new displaykey, then you will also need to reload PCF files using ALT+SHIFT+L, the Guidewire API and/or internal server tools.

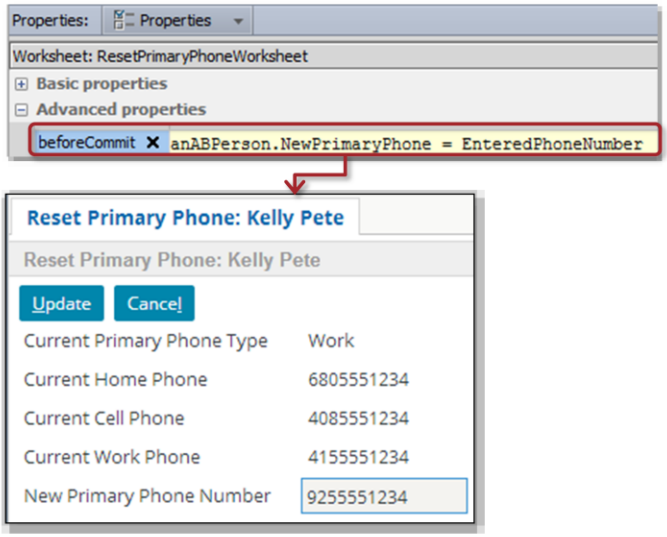
Step 4: Reference enhancement (1)



- Reference getters using the same syntax as fields and other derived properties
- Value property of input widget references the getter
- Age getter specifies the value of the Age widget

Entity enhancement elements are referenced using the same syntax as base application entity elements. In the slide example, ABPersonEnhancement.gsx defines the Age property. The getter calculates the age using the date of birth date value.

Step 4: Reference enhancement (2)

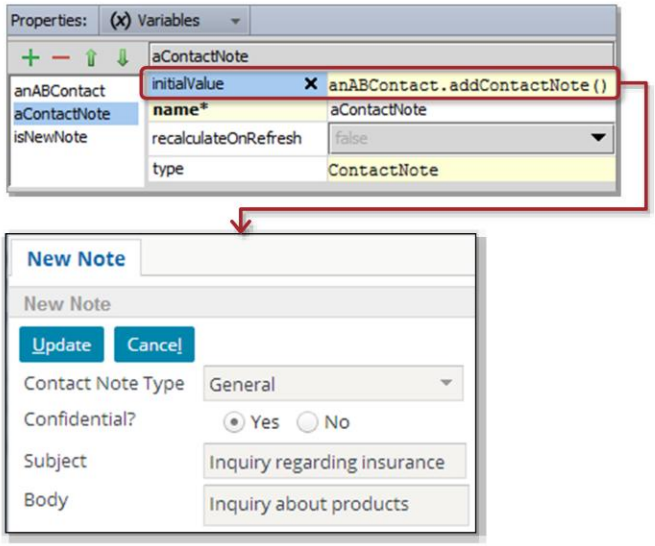


- Reference setters using the same syntax as fields

- beforeCommit property of the worksheet location references the NewPrimaryPhone setter

Entity enhancement elements are referenced using the same syntax as base application entity elements. In the slide example, ABPersonEnhancement.gsx defines the NewPrimaryPhone setter enhancement property. The setter updates the number per the type PrimaryPhone type key prior to committing the data to the database.

Step 4: Reference enhancement (3)



- Reference methods using the same syntax as object methods

- Worksheet location variable initialValue property creates a new Contact Note using the addContactNote() method

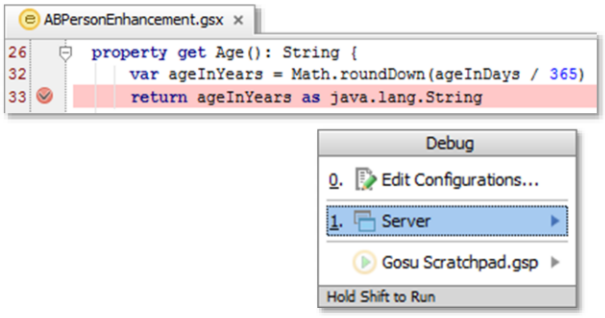
Entity enhancement elements are referenced using the same syntax as base application entity elements. The addContactNote method is used to create a new, empty contact note related to anABContact. You can find the method referenced in ContactNoteWorksheet.pcf

Lesson outline

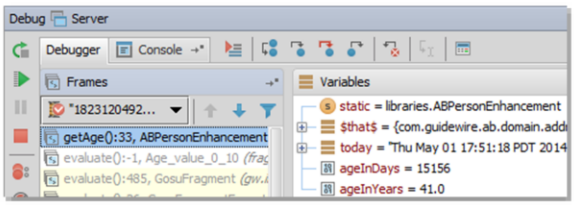
- Entity enhancement fundamentals
- Working with entity enhancements
- Debugging enhancements
- Gosu enhancements for Java

Steps to debug enhancements

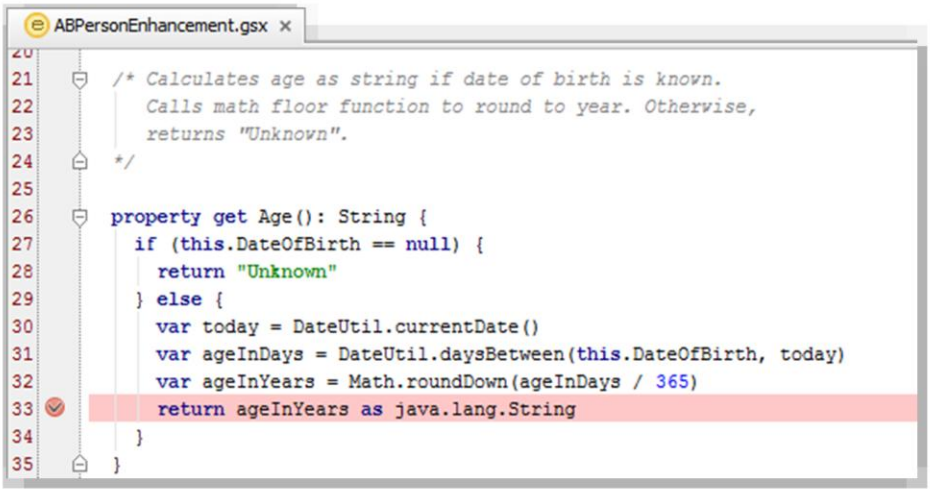
- Set breakpoints in enhancement Debug Server
  - ALT+SHIFT+F9
  - Select Server
- Console tab
  - Verify output reads
- Perform actions to hit breakpoint
- In Debug tool window, review Debugger
- Resume program
  - F9




```
*** AppName ready ***
```



Breakpoints

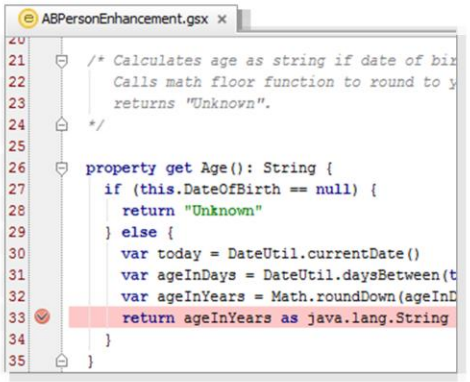


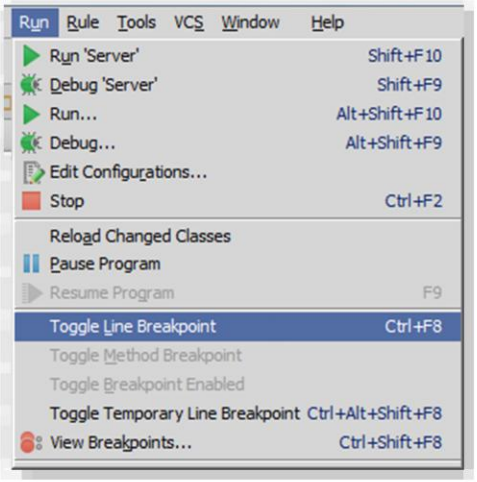
- A **breakpoint** indicates a place where you want to suspend the execution of Gosu code

Because you can step through lines, you normally need only one breakpoint for each section of code you want to investigate.

Setting breakpoints

- Click in the gutter for the given line
- Select line
  - Main menu → Run → Toggle Line Breakpoint
  - CTRL+F8

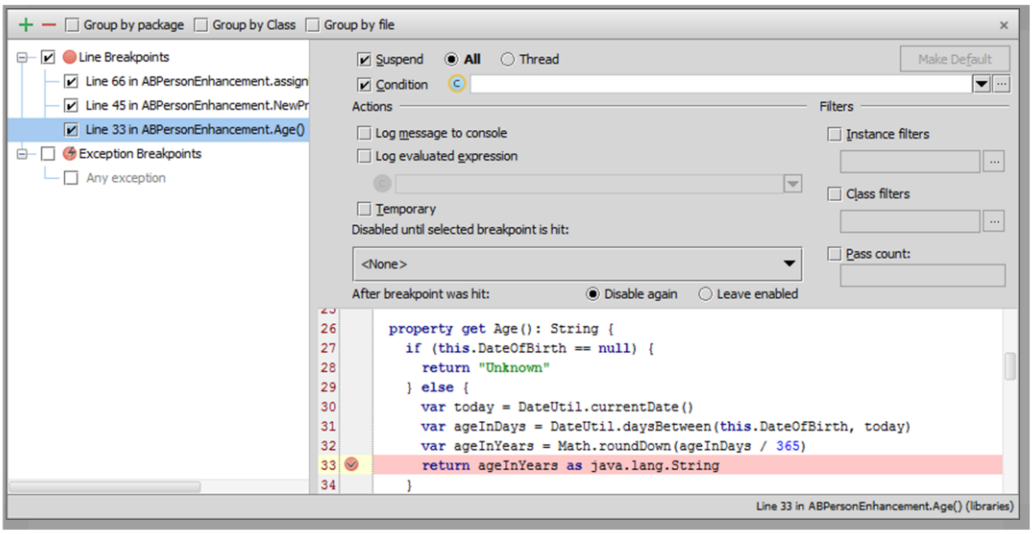




You can set a breakpoint by clicking in gutter for the given line in the editor. You can also select the line and then select Main menu → Run → Toggle Line Breakpoint or use the CTRL+F8 keystroke.

View all breakpoints

- Run → View Breakpoints… or CTRL+SHIFT+F8
  - Enable, disable, define action and conditions



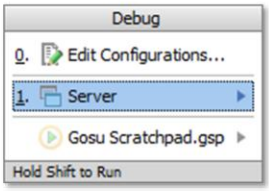
In addition to CTRL+SHIFT+F8, right-clicking on the breakpoint also displays a menu with choices to Edit, Disable, Remove, or View Breakpoints. In the slide example, the View Breakpoints dialog shows all the breakpoint options.

The Breakpoint options dialog allows you to review all breakpoints. In the Toolbar, you can add a breakpoint, remove a breakpoint, display breakpoints under their respective packages rather than under their types, display breakpoints under their respective classes, and display breakpoints under their respective files. There are various Breakpoint options to set. You can enable a suspend policy for a breakpoint. For example, for a Thread policy the thread where the breakpoint is hit is suspended. You can also specify a condition in either Gosu or Java. You can also specify actions, such as logging the evaluated expression.

Debug 'Server'

- Start debug Server
  - ALT+SHIFT+F9
  - Select Server
- Console tab
  - Verify output reads

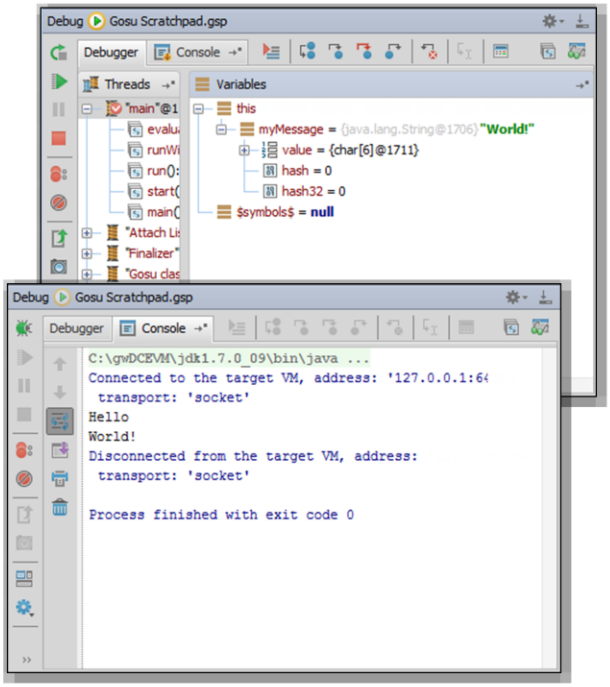
```
*** <GuidewireApplication> ready ***
```



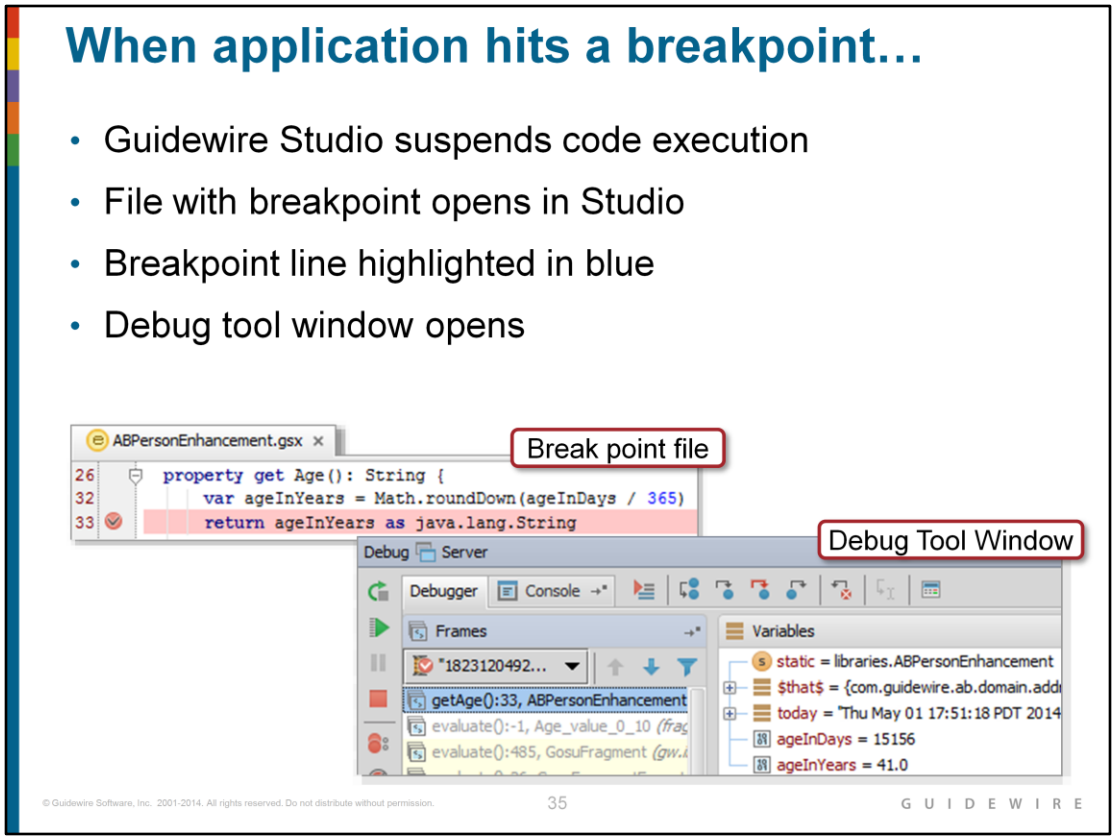
To debug the server, select Main menu → Run → Debug 'Server' or Main menu → Run → Debug… → Server. In the Debug tools window, confirm that the application is running and is ready (***** AppName ready *****).

Debug tool window

- ALT+5
  - Opens Debug window
- Debugger tab
  - Rerun, Resume, Pause, Stop
  - View breakpoints
  - Step over, into and out
  - Inspect and watch
- Console tab
  - View output



The debugger enables you to execute your application step by step, examine program information related to variables, watches, or threads, and change your program without leaving Guidewire Studio.



To hit breakpoints, you must run the application in a debug 'server' process.

Because you can step through lines, you normally need only one breakpoint for each section of code you want to investigate. There is no advantage to having breakpoints on multiple consecutive lines because the first breakpoint suspends normal execution and normal execution does not resume until you request it to.

The Frame tab lists the object passed to the code (for enhancements, this is the root entity or class). Variable values at the breakpoint are visible in the Variables window. Frames shows the sequence of actions executed until the breakpoint.

Stepping through code

- Several step tools are available to help in debugging


Step over (F8) advances you to the next line in the file

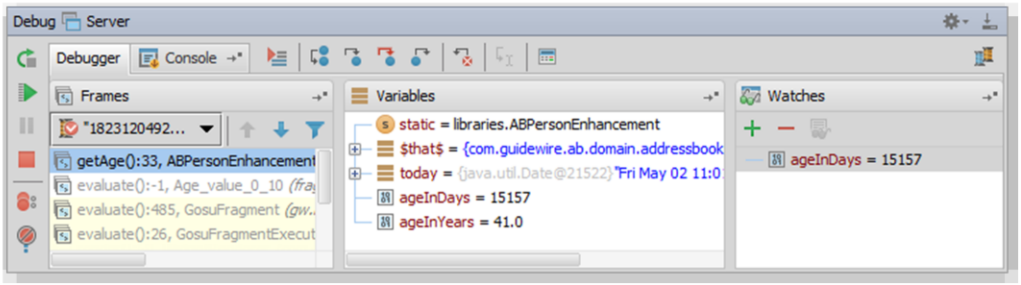

Step into (F7) advances you to the next line executed


Force step into (Alt+Shift+F7) steps into, ignore stepping filters for libraries, constructors, etc.


Step out(Shift+F8) step to the first line executed after returning from this method

Once the debugger pauses execution, you can step through the code in one of several ways. The most basic is to stepping through code one line at a time using Step Over (F8).

Debugger



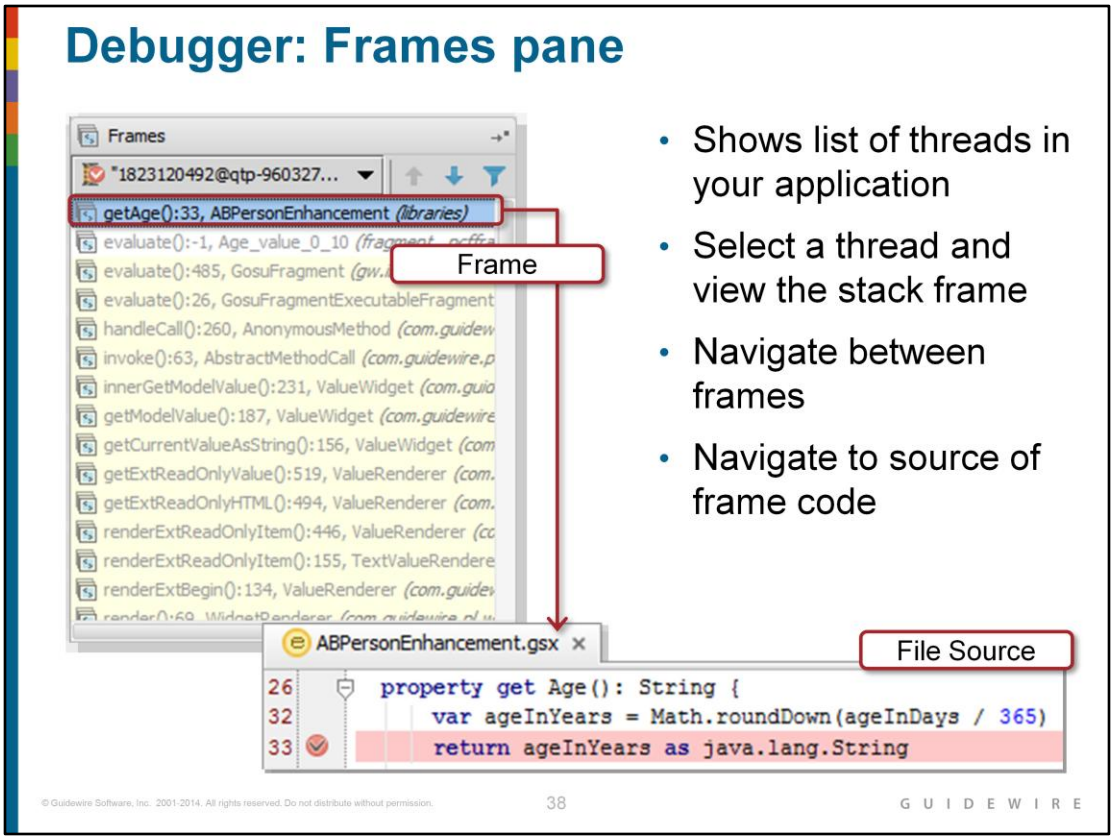
- For breakpoint, debugger shows how code executes
- Debug Gosu code running in the Debug 'Server' process

Class

Enhancement

Rule

When the Debug 'Server' process hits a breakpoint in Gosu code, the debugger suspends the normal execution of code until you resume it. When you resume the debugger, code executes as normal until the debugger hits another breakpoint.



The Frames pane enables you to gain access to the list of threads running in your application, export threads to a text file, and customize the thread presentation.

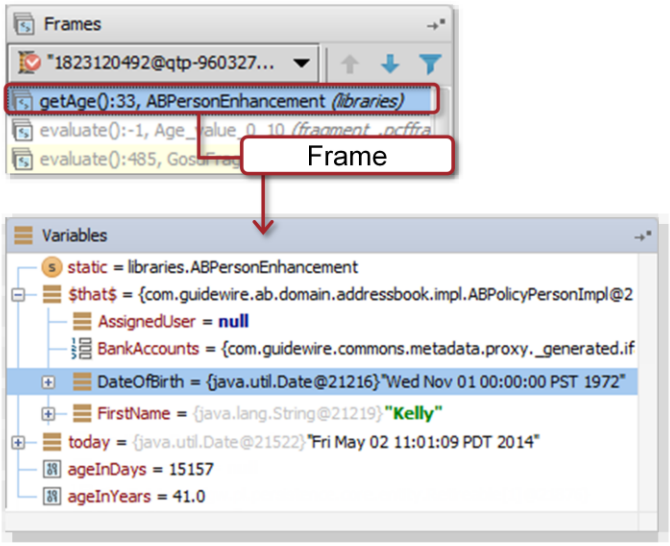
For each thread, you can view the stack frame, examine frames, navigate between frames, and automatically jump to the source code of a frame in the editor.

A thread can be chosen via a thread selector drop-down list on top of the pane. The status and type of a thread is indicated by the special icon and textual note next to the thread's name.

To examine the values stored in a frame, use the Variables pane of the Debug tool window.

Debugger: Variables pane

- Examine variables, fields, arrays, primitive types, and object of a selected frame



- Enable Enhance Entities Visualization to view all entity properties
  - Project Settings → IDE Settings → Guidewire Studio → Debugger Settings
- Add variable to watches

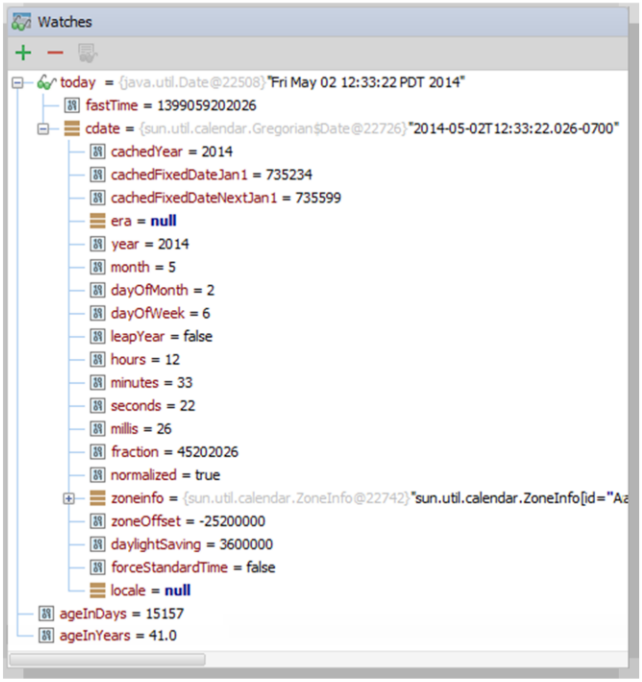
GUIDEWIRE

When a stack frame is selected in the Frames pane, the Variables pane displays all the data within its scope (method parameters, local and instance variables). In the variables pane you can set labels for the objects, inspect objects, evaluate expressions, add variables to watches and more. You can examine static variables, fields, arrays, primitive types, and objects.

Enable Enhance Entities Visualization in Guidewire Studio to view all entity properties. In Project Settings, in IDE Settings, select Guidewire Studio. Then, in Debugger Settings, check the Enhance Entities Visualization checkbox.

Debugger: Watches pane

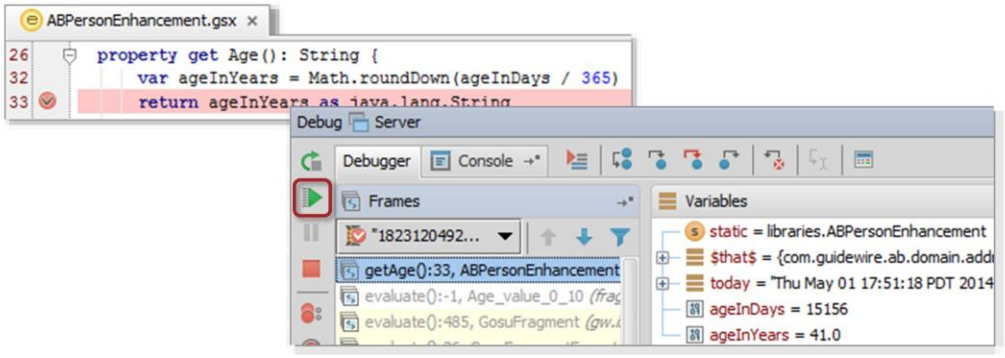
- A **watch** is an expression whose value you wish to observe
  - Click Add → Enter property or expression
  - Drag object or property from the Variable pane to watches pane
  - Drag variable or property from enhancement to watches pane



In the Watches pane you can evaluate any number of variables or expressions in the context of the current stack frame. The values are updated with each step through the application and become visible every time the application is suspended.

The easiest way to populate the Watches pane is to run the code to a desired point, then drag the object or property to the Watches pane.

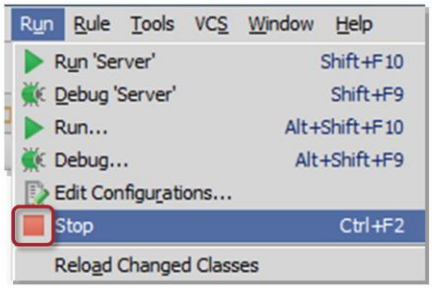
Resume debugging

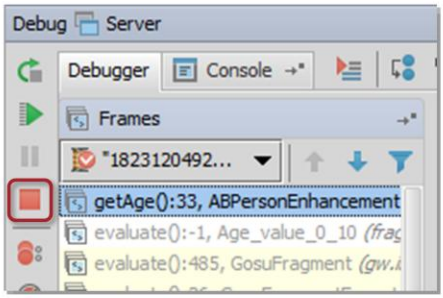


- To resume the execution of code…
  - Main menu → Run → Resume
  - F9
  - Or, in the Debug Tool Window, click Resume
- Code continues to hitting next breakpoint

Stopping debug server

- To stop the debug server process…
  - Main menu → Run → Stop
  - CTRL+F2
  - Or, in the Debug Tool Window, click Stop





Lesson outline

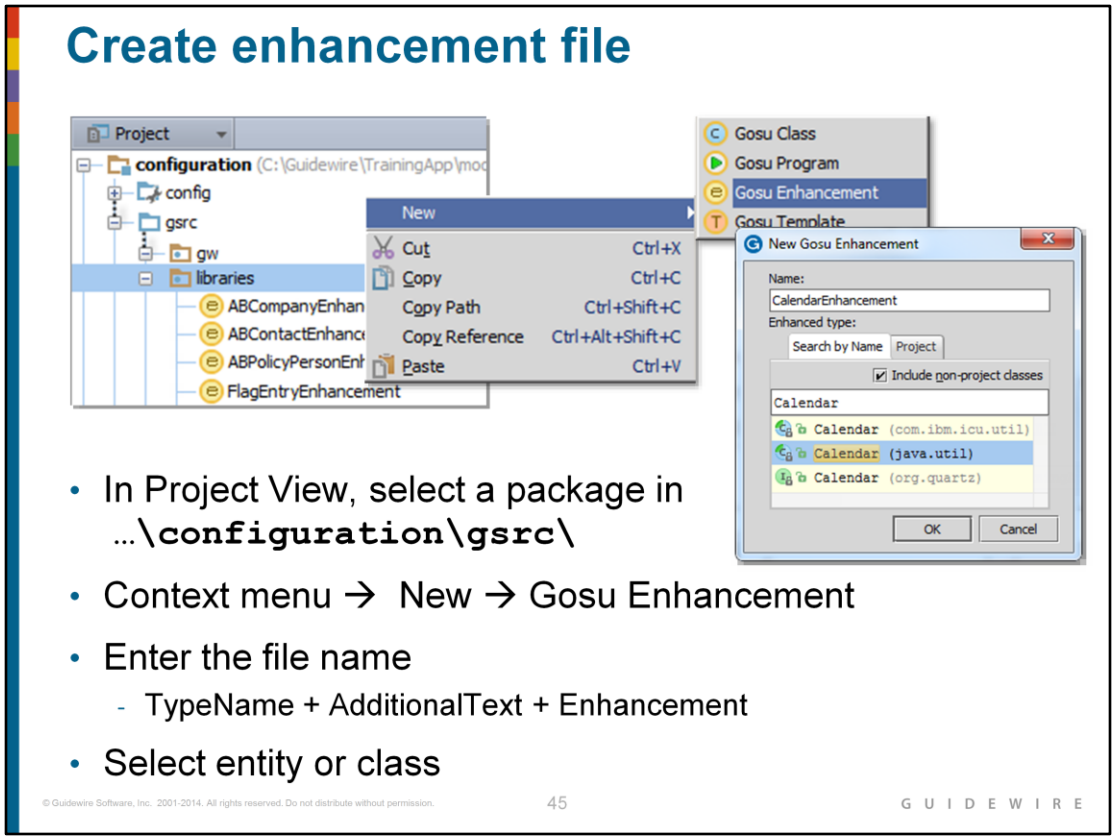
- Entity enhancement fundamentals
- Working with entity enhancements
- Debugging enhancements
- Gosu enhancements for Java

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`**
- Configuration resources often enhance Java classes and interfaces with static methods
- Deploy and debug the same way as other enhancements

You call static methods directly from the class itself as opposed to creating an instance of the class.



Guidewire recommends that you create an enhancement in a custom package named after your company. For example, if you are implementing an application for Acme Insurance, then you would create all custom enhancements in a package named Acme or in sub-packages of an Acme package.

The recommended naming convention for Java Enhancements is <TypeName> + <AdditionalTextWhenNeeded> + Enhancement. In some cases, you may need to have multiple enhancements for a single class. In this case, the <AdditionalTextWhenNeeded> should be used to clarify the difference between the enhancements.

Create static methods

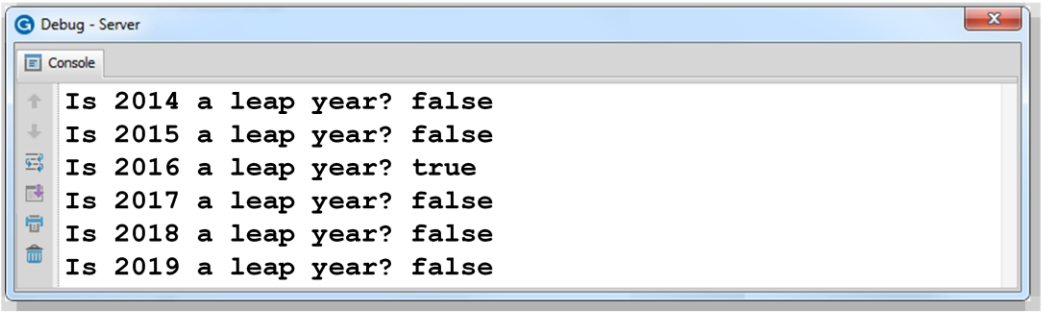
```
 1  package libraries
 2
 3  uses gw.api.util.DateUtil
 4
 5  enhancement CalendarEnhancement: java.util.Calendar {
…9
10    static function isLeapYear(yearToCheck: int): boolean {
11      var FirstDate = "01/01/" + yearToCheck
12      var SecondDate = "01/01/" + ( yearToCheck + 1 )
13      var DaysBetween = DateUtil.daysBetween(
                          FirstDate as java.util.Date,
                          SecondDate as java.util.Date)
14      return (DaysBetween == 366)
15    }
16  }
```

- Line 5: imports enhancement of the Java Calendar class
- Line 10: callsisLeapYear() static function

Static methods are not called from instances. Because static methods do not work off an instance, there is no need use the this keyword in a static Gosu enhancement method of a Java class.

Reference static methods

```
1  uses java.util.Calendar
2  var output = ""
3  for (i in 2014..2019) {
4    var result = Calendar.isLeapYear(i)
5    output += String.format("Is %s a leap year? %s \n",{i, result})
6  }
7  print(output)
```



- Line 1: Declares full qualified name for Calendar class
- Line 4: References the enhancement static function

In the slide example, Gosu Scratchpad executes code that references the static method. The code outputs to the console in Studio.
Line 3 declares a loop that iterates through a range of values and instantiates an iterator variable (i).
Line 4 references the isLeapYear() enhancement function for the Java Calendar class. The iterator variable (i) is the argument value for the isLeapYear() static function.
Line 5 concatenates a string using the String.format() function. The function creates a string using the iterator value and the boolean variable.
Line 7 outputs the string to the console.

Lesson objectives review

- You should now be able to:
  - Describe the purpose and functionality of Gosu enhancements
  - Create entity enhancements
  - Reference entity enhancement properties and methods
  - Create Gosu enhancements for Java classes and interaces
  - Debug enhancements

Review questions

1. Where can you create enhancements in a project?
2. What type of logic does a getter implement? A setter? A method?
3. When you create a new enhancement, what code does Studio add for you automatically?
4. For an enhancement on the ABContact entity, what code would you write to reference the given ABContact's AssignedUser field?
5. How do you create an enhancement method that returns no value?
6. How do you reference enhancement properties and methods?

Answers
1) Create enhancements in an Guidewire application project in Guidewire Studio in …\configuration\gsrc\.
2) 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.
3) Studio automatically creates the package statement and the enhancement declaration statement and the type reference.
4) this.AssignedUser
5) You would create the method and optionally specify a return type of void.
6) You reference enhancement properties and methods using the same syntax as base application entity properties and methods: object.propertyOrMethodName.

Notices