Skip Headers

Oracle® Fusion Applications CRM Extensibility Guide
11*g* Release 7 (11.1.7)
Part Number E20388-06

Home Contents Book List Contact Us

Previous Next

PDF

6 Application Composer: Using Groovy Scripts

This chapter contains the following:

Using Groovy Scripts : Overview

Read this chapter to learn about how and where you can use Groovy scripting in Oracle Fusion CRM Application Composer.

> **Important**
>
> To fully understand all the scripting features available to you in Application Composer, you should also review the Oracle Fusion CRM Application Composer Scripting Guide (ID 1354807.1) on My Oracle Support at https://support.oracle.com.

In this chapter, you will learn about:

- Where you can use Groovy in your application, along with examples of one or more lines of Groovy code
- How to access view objects using the newView() function, for programmatic access to object data
- How to create global functions, which is code that multiple objects can share
- How to call Web services from your Groovy scripts. You might call a Web service for access to internal or external data, or for example, to perform a calculation on your data.

You write Groovy scripts using Application Composer's expression builder, which appears in many places throughout Application Composer as you customize existing objects or create new custom ones.

- You will write shorter scripts to provide an expression to calculate a custom formula field's value or to calculate a custom field's default value, for example.
- You will generally write somewhat longer scripts to define a field-level validation rule or an object-level validation rule, for example.

Additional examples of where you write Groovy scripts in Application Composer are described in "Groovy Scripting: Explained."

To learn more about how to best utilize the features available in the expression builder when writing scripts, see "Groovy Tips and Techniques" in the Oracle Fusion CRM Application Composer Scripting Guide (ID 1354807.1) on My Oracle Support at https://support.oracle.com.

Groovy Scripting : Explained

Groovy is a standard, dynamic scripting language for the Java platform for which the Oracle Fusion CRM Application Composer provides deep support. This topic provides an overview of where you can use groovy in your application and gives some samples of one or more lines of Groovy code. You can also find information on Supported Classes and Methods for Use in Groovy Scripts and some examples in the Related Topics section.

For more information on groovy scripts, see Oracle Fusion CRM Application Composer Scripting Guide (ID 1354807.1) on My Oracle Support at https://support.oracle.com.

Note

Read "Supported Classes and Methods for Use in Groovy Scripts", which documents the only classes and methods you may use in your Groovy scripts. Using any other class or method will raise a security violation error when you migrate your code to later Oracle Fusion CRM maintenance releases. Therefore, we strongly suggest that the Groovy code you write uses only the classes and methods shown there to avoid the unpleasant and possibly time-consuming task of having to rewrite your code in the future.

Terminology Explained

Throughout the document the term script is used to describe one or more lines of Groovy code that the Oracle ADF framework executes at run time. Often a very-short script is all that is required.

For example, to validate that a CommissionPercentage field's value does not exceed 40%, you might use a one-line script like:

```
return CommissionPercentage < 0.40
```

In fact, this one-liner can be conveniently shortened by dropping the return keyword since the return keyword is always implied on the last line of a script:

```
CommissionPercentage < 0.40
```

For slightly more complicated logic, your script might require some conditional handling. For example, suppose the maximum commission percentage is 40% if the salesperson's job grade is less than or equal to 3, but 60% if the job grade is higher. Your script would grow a little to look like this:

```
if (JobGrade <= 3) {
  return CommissionPercentage < 0.40
}
else {
  return CommissionPercentage < 0.60
}
```

Scripts that you'll write for other purposes like complex validation rules or reusable functions may span multiple pages, depending on your needs.

When a context requiring a Groovy script will typically use a short (often, one-line) script, we emphasize that fact by calling it an expression, however technically the terms script and expression are interchangeable. Anywhere you can provide a one-line expression is also a valid context for providing a multi-line script if the need arises. Whether you provide a short expression or a multi-line script, the syntax and features at your disposal are the same. You need only pay attention that your code returns a value of the appropriate type for the context in which you use it.

The Groovy Scripting: Examples topic includes all the return types. This topic highlights the expected return type for each script example.

Where You will Use Groovy in Your Application

There are a number of different contexts where you will use Groovy scripts as you customize existing objects or create new custom ones.

You will write shorter scripts to provide an expression to:

- calculate a custom formula field's value
- calculate a custom field's default value
- make a custom field conditionally updateable, or
- make a custom field conditionally required
- define the condition for executing an object workflow

You will generally write somewhat longer scripts to define:

- a field-level validation rule
- an object-level validation rule
- a trigger to complement default processing
- utility code in a global function, or
- reusable behavior in an object function

If you anticipate calling the same code from multiple different contexts, any of your scripts can call the reusable code you write in either global functions or object functions. As their name implies, global functions

can be called from scripts in any object or from other global functions. Object functions can be called by any scripts in the same object, or even triggered by a button in the user interface.

After exploring the Groovy basic techniques needed to understand the examples, documented in Oracle Fusion CRM Application Composer Scripting Guide (ID 1354807.1) on My Oracle Support at https://support.oracle.com, see "Groovy Scripting: Examples" for a concrete example of each of these usages. The Oracle Fusion CRM Application Composer Scripting Guide also includes a section on "Groovy Tips and Techniques" for additional tips and techniques on getting the most out of Groovy in your application.

Groovy Scripting : Examples

The following examples show how you can use Groovy in all of the different supported contexts in your application.

Providing an Expression to Calculate a Custom Formula Field's Value

When you need a calculated field or a transient value-holder field with an optional initial, calculated value, use a formula field.

1. For read-only calculated fields:

A formula field defaults to being a read-only, calculated value. It displays the value resulting from the run time evaluation of the calculation expression you supply. By using the **Depends On** multi-select list in the field create or edit page, you can configure the names of fields on which your expression depends. By doing this, its calculated value will update dynamically when any of those **Depends On** fields' value changes. The expected return type of the formula field's expression must be compatible with the formula field type you specified (Number, Date, or Text).

For example, consider a custom **TroubleTicket** object. If you add a formula field named **DaysOpen**, you can provide its calculated value with the expression:

```
(today() - CreationDate) as Integer /* truncate to whole number of days */
```

2. For transient value holder fields with optional calculated initial value:

If you want to allow the end user to override the calculated value, then mark your formula to be updateable. An updateable formula field is a "transient value holder" whose expression provides the value of the field until the user overrides it. If the user overrides the value, the object remembers this user-entered value for the duration of the current transaction so that your validation rules and triggers can reference it. If you have configured one or more **Depends On** fields for your updateable formula field, then note that the value of the formula will revert back to the calculated value should any of the dependent fields' value change. If you want a transient field whose initial value is null until the user fills it in, simply provide no formula expression for your updateable formula field to achieve this.

Providing an Expression to Calculate a Custom Field's Default Value

When a new row is created for an object, the value of a custom field defaults to **null** unless you configure a default value for it. You can supply a literal default value of appropriate type or supply an expression to

calculate the default value for new rows. The default value expression is evaluated at the time the new row is created. The expected return type of your field's default value expression must be compatible with the field's type (Number, Date, Text, and so on).

For example, consider a custom **CallbackDate** field in a **TroubleTicket** object. If you want the callback back for a new Help Request to default to 3 days after it was created, then you can provide a default expression of:

```
CreationDate + 3
```

Providing an Expression to Make a Custom Field Conditionally Updateable

1. To provide an expression to make a custom field conditionally updateable:

A custom field can be updateable or read-only. By default, any non-formula field is updateable. Alternatively, you can configure a conditionally updateable expression. If you do this, it is evaluated each time a page displaying the field is rendered or refreshed. The expected return type of the expression is **boolean**. If you define one for a field, you must also configure the **Depends On** list to indicate the names of any fields on which your conditionally updateable expression depends. By doing this, your conditionally updateable field will interactively enable or disable as appropriate when the user changes the values of fields on which the conditionally updateable expression depends.

For example, consider a custom **TroubleTicket** object with **Status** and **Justification** fields. Assume you want to prevent a user from editing the justification of a closed Help Request. To achieve this, configure the conditionally updateable expression for the **Justification** field as follows:

```
Status_c != 'Closed'
```

After configuring this expression, you must then indicate that the **Justification** field depends on the **Status** field as described in "Understanding When to Configure Field Dependencies", in Oracle Fusion CRM Application Composer Scripting Guide (ID 1354807.1) on My Oracle Support at https://support.oracle.com. This ensures that if a Help Request is closed during the current transaction, or if a closed Help Request is reopened, that the **Justification** field becomes enabled or disabled as appropriate.

Tip

A field configured with a conditionally updateable expression only enforces the conditional updateability through the Web user interface. For more information on how to ensure it gets enforced for Web service access as well, see "Enforcing Conditional Updateability of Custom Fields for Web Service Access" in Oracle Fusion CRM Application Composer Scripting Guide (ID 1354807.1) on My Oracle Support at http://www.oracle.com/technetwork/indexes/documentation.

---

2. To provide an expression to make a custom field conditionally required:

A custom field can be optional or required. By default it is optional. Alternatively, you can configure a conditionally required expression. If you do this, it is evaluated each time a page displaying the field is rendered or refreshed, as well as when the object is validated. The expected return type of the expression is **boolean**. If you define one for a field, you must also configure the **Depends On** list to indicate the names of any fields on which your conditionally required expression depends. By doing this, your conditionally required field will interactively show or hide the visual indicator of the field's being required as appropriate when the user changes the values of fields on which the conditionally required expression depends.

For example, consider a custom **TroubleTicket** object with **Priority** and **Justification** fields. Assume that priority is an integer from 1 to 5 with priority 1 being the most critical kind of problem to resolve. To enforce that a justification is required for Help Requests whose priority is 1 or 2, configure the conditionally required expression for the **Justification** field as follows:

```
Priority_c <= 2
```

After configuring this expression, you must then indicate that the **Justification** field depends on the **Priority** field as described in "Understanding When to Configure Field Dependencies", in Oracle Fusion CRM Application Composer Scripting Guide (ID 1354807.1). This ensures that if a Help Request is created with priority 2, or an existing Help Request is updated to increase the priority from 3 to 2, the **Justification** field becomes mandatory.

Defining a Field-Level Validation Rule

1. To define a field-level validation rule:

A field-level validation rule is a constraint you can define on any standard or custom field. It is evaluated whenever the corresponding field's value is set. When the rule executes, the field's value has not been assigned yet and your rule acts as a gatekeeper to its successful assignment. The expression (or longer script) you write must return a **boolean** value that indicates whether the value is valid. If the rule returns **true**, then the field assignment will succeed so long as all other field-level rules on the same field also return **true**. If the rule returns **false**, then this prevents the field assignment from occurring, the invalid field is visually highlighted in the UI, and the configured error message is displayed to the end user. Since the assignment fails in this situation, the field retains its current value (possibly **null**, if the value was **null** before), however the UI component in the Web page allows the user to see and correct their invalid entry to try again. Your script can use the **newValue** keyword to reference the new value that will be assigned if validation passes. To reference the existing field value, use the **oldValue** keyword. A field-level rule is appropriate when the rule to enforce only depends on the new value being set. You can use the Keywords tab of the Expression Palette to insert the **newValue** and **oldValue** keywords.

For example, consider a custom **TroubleTicket** object with a **Priority** field. To validate that the number entered is between 1 and 5, your field-level validation rule would look like this:

Field-Level Validation Rule Component	Value
Field Name	Priority
Rule Name	Validate_Priority_Range
Rule Body	`newValue == null || (1..5).contains(newValue as Integer)`
Error Message	The priority must be in the range from 1 to 5.

Tip

If a validation rule for field **A** depends on the values of one or more other fields (For example, **Y** and **Z**), then create an object-level rule and programmatically signal which field or fields should be highlighted as invalid to the user, as explained in "Setting Invalid Fields for the UI in an Object-Level Validation Rule" in Oracle Fusion CRM Application Composer Scripting Guide (ID 1354807.1) on My Oracle Support at https://support.oracle.com.

2. To define an object-level validation rule:

An object-level validation rule is a constraint you can define on any standard or custom object. It is evaluated whenever the framework attempts to validate the object. This can occur upon submitting changes in a web form, when navigating from one row to another, as well as when changes to an object are saved. Use object-level rules to enforce conditions that depend on two or more fields in the object. This ensures that regardless of the order in which the user assigns the values, the rule will be consistently enforced. The expression (or longer script) you write must return a **boolean** value that indicates whether the object is valid. If the rule returns **true**, then the object validation will succeed so long as all other object-level rules on the same object return **true**. If the rule returns **false**, then this prevents the object from being saved, and the configured error message is displayed to the end user.

For example, consider a **TroubleTicket** object with **Priority** and **AssignedTo** fields, where the latter is a dynamic choice list field referencing **Contact** objects whose **Type** field is a **Staff Member**. To validate that a Help Request of priority 1 or 2 cannot be saved without being assigned to a staff member, your object-level rule would look like this:

Object-Level Validation Rule Component	**Value**
Rule Name	Validate_High_Priority_Ticket_Has_Owner

Rule Body	`// Rule depends on two fields, so must be written as object-level rule`
	`if (Priority_c <= 2 && AssignedTo_Id_c == null) {`
	`  // Signal to highlight the AssignedTo field on the UI as being in error`
	`  adf.error.addAttribute('AssignedTo_c')`
	`  return false;`
	`}`
	`return true;`

Error Message	A Help Request of priority 1 or 2 must have a staff member assigned to it.

Defining Utility Code in a Global Function

Global functions are useful for code that multiple objects want to share. To call a global function, preface the function name with the adf.util. prefix. When defining a function, you specify a return value and can optionally specify one or more typed parameters that the caller will be required to pass in when invoked. The most common types for function return values and parameters are the following:

- **String**: a text value
- **Boolean**: a logical true or false value
-
- **BigInteger**: an integer of arbitrary precision
-
- **BigDecimal**: a decimal number of arbitrary precision
- **Date**: a date value with optional time component
- **List**: an ordered collection of objects
- **Map**: an unordered collection of name/value pairs

- **Object**: any object

In addition, a function can define a **void** return type which indicates that it returns no value.

> **Note**
>
> A global function has no current object context. To write global functions that work on a particular object, refer to "Passing the Current Object to a Global Function," in Oracle Fusion CRM Application Composer Scripting Guide (ID 1354807.1) on My Oracle Support at http://www.oracle.com/technetwork/indexes/documentation.

For example, you can create the following two global functions to define standard helper routines to log the start of a block of Groovy script and to log a diagnostic message. The examples in Oracle Fusion CRM Application Composer Scripting Guide (ID 1354807.1) on My Oracle Support at http://www.oracle.com/technetwork/indexes/documentation, make use of them.

This table describes how to set up a global function to log the start of a block of Groovy script.

Global Function Component	**Value**
Function Name	logStart
Return Type	void
Parameters	• Name: scriptName • Type: String
Function Definition	`// Log the name of the script` `println("[In: ${scriptName}]")`

This table describes how to set up a global function to log a diagnostic message.

Global Function Component	**Value**
Function Name	log
Return Type	void
Parameters	• Name: message • Type: String
Function Definition	`// Log the message, could add other info` `println(message)`

Defining Reusable Behavior with an Object Function

Object functions are useful for code that encapsulates business logic specific to a given object. You can call object functions by name from any other script code related to the same object. In addition, you can invoke them using a button or link in the user interface. The supported return types and optional parameter types are the same as for global functions (described above).

For example, you might define the following **updateOpenTroubleTicketCount()** object function on a **Contact** custom object. It begins by calling the **logStart()** global function above to log a diagnostic message in a standard format to signal the beginning of a block of custom Groovy script. It calls the **newView()** built-in function (described in "Accessing the View Object for Programmatic Access to Business Objects" in Oracle Fusion CRM Application Composer Scripting Guide (ID 1354807.1) on My Oracle Support) to access the view object for programmatic access of Help Requests, then calls another global function **applyFilter()** (described in "Simplifying Most Common View Criteria Creation with a Global Function" in Oracle Fusion CRM Application Composer Scripting Guide (ID 1354807.1)) to apply a filter to find Help Requests related to the current contact's id and having either **Working** or **Waiting** as their current status. Finally, it calls **getEstimatedRowCount()** to retrieve the count of Help Requests that qualify for the filter criteria.

This table describes the updateOpenTroubleTicketCount() object function on a Contact custom object:

Object Function Component	**Value**
Function Name	updateOpenTroubleTicketCount
Return Type	void
Parameters	None

Function Definition	`adf.util.logStart('updateOpenTroubleTicketCount')`
	`// Access the view object for TroubleTicket programmatic access`
	`def tickets = newView('TroubleTicket_c')`
	`// Use a global function to apply a query filter to Help Request`
	`// view object where Contact_Id = Id of current Contact and`
	`// Status is either 'Working' or 'Waiting'`
	`adf.util.applyFilter(tickets,[Status_c:['Working','Waiting'],`
	`Contact_Id__c:Id])`
	`// Update OpenTroubleTickets field value`
	`setAttribute('OpenTroubleTickets_c',tickets.getEstimatedRowCount())`

Defining an Object Level Trigger to Complement Default Processing

Triggers are scripts that you can write to complement the default processing logic for a standard or custom object. You can define triggers both at the object-level and the field-level. The following object level triggers are available:

- **After Create**: Fires when a new instance of an object is created. Use to assign programmatic default values to one or more fields in the object.
- **Before Modify**: Fires when the end-user first modifies a persistent field in an existing, queried row.
- **Before Invalidate**: Fires on an existing object when its first persistent field is modified. Fires on a valid parent object when a child row is created, removed, or modified.
- **Before Remove**: Fires when an attempt is made to delete an object. Returning false stops the row from being deleted and displays the optional trigger error message.
- **Before Insert in Database**: Fires before a new object is inserted into the database.

  See tip below.
- **After Insert in Database**: Fires after a new object is inserted into the database.

  See tip below.

- **Before Update in Database**: Fires before an existing object is modified in the database.

  See tip below.

- **After Update in Database**: Fires after an existing object is modified in the database.

  See tip below.

- **Before Delete in Database**: Fires before an existing object is deleted from the database.

- **After Delete in Database**: Fires after an existing object is deleted from the database.

- **After Changes Posted to Database**: Fires after all changes have been posted to the database, but before they are permanently committed. Can be used to make additional changes that will be saved as part of the current transaction.

  See tip below.

- **Before Commit in Database**: Fires before the change pending for the current object (insert, update, delete) is made permanent in the current transaction. Any changes made in this trigger will not be part of the current transaction. Use "After Changed Posted to Database" trigger if your trigger needs to make changes.

- **After Commit in Database**: Fires after the change pending for the current object (insert, update, delete) is made permanent in the current transaction.

- **Before Rollback in Database**: Fires before the change pending for the current object (insert, update, delete) is rolled back

- **After Rollback in Database**: Fires after the change pending for the current object (insert, update, delete) is rolled back

Consider a Contact object with a **OpenTroubleTickets** field that needs to be updated any time a Help Request is created or modified. You can create the following two triggers on the **TroubleTicket** object which both invoke the **updateOpenTroubleTicketCount()** object function described above.

This table describes how to set up a trigger on the **TroubleTicket** object.

Trigger Component	**Value**
Trigger Object	TroubleTicket
Trigger	After Insert In Database
Trigger Name	After_Insert_Set_Open_Trouble_Tickets

Trigger Definition	`adf.util.logStart('After_Insert_Set_Open_Trouble_Tickets')`
	`// Get the related contact for this Help Request`
	`def relatedContact = Contact_Obj_c`
	`// Update its OpenTroubleTickets field value`
	`relatedContact?.updateOpenTroubleTicketCount()`

This table describes how to set up a second trigger on the **TroubleTicket** object.

Trigger Component	**Value**
Trigger Object	TroubleTicket

Trigger	After Update In Database
Trigger Name	After_Update_Set_Open_Trouble_Tickets

Trigger Definition	`// Get the related contact for this Help Request`
	`def relatedContact = Contact_Obj_c`
	`// Update its OpenTroubleTickets field value`
	`relatedContact?.updateOpenTroubleTicketCount()`

Tip

When setting the value of an attribute using the triggers listed below, check to see if the attribute being set already has the value being assigned, and if so, skip performing the assignment.

- Before Insert in Database
- After Insert in Database
- Before Update in Database
- After Update in Database
- After Changes Posted to Database

For example:

```
if (Text_c != 'AfterCommit') {
            setAttribute('Text_c', "AfterCommit")
        }
```

Defining a Field-Level Trigger to React to Value Changes

Field-level triggers are scripts that you can write to complement the default processing logic for a standard or custom field.

The After Field Changed trigger fires when the value of the related field has changed (implying thatit has passed any field-level validation rules that might be present). Use the After Field Changed trigger to calculate other derived field values when another field changes value. Do not use a field-level validation rule to achieve this purpose because while your field-level validation rule may succeed, other field-level validation rules may fail and stop the field's value from actually being changed. Since generally you only want your field-change derivation logic to run when the field's value actually changes, the After Field Changed trigger guarantees that you get this desired behavior.

Supported Classes and Methods for Use in Groovy Scripts: Explained

Groovy is a standard, dynamic scripting language for the Java platform for which the Oracle Fusion CRM Application Composer provides deep support. This topic covers the supported classes and methods for use in Groovy scripts.

Classes and Methods

When writing Groovy scripts, you may only use the classes and methods that are documented in the table below. Using any other class or method may work initially, but will throw a run time exception when you migrate your code to later versions. Therefore, we strongly suggest that you ensure the Groovy code you write adheres to the classes and methods shown here. For each class, in addition to the method names listed in the table, the following method names are also allowed:

- equals()
- hashCode()
- toString()

In contrast, the following methods are never allowed on any object:

- finalize()
- getClass()
- getMetaClass()
- notify()
- notifyAll()
- wait()

Class Name	Allowed Methods	Package
ADFContext	getLocale() getSecurityContext() getUserRoles() isUserInRole()	oracle.adf.share
Array	Any constructor Any method	java.sql
Array	getArray() getElemType() getList()	oracle.jbo.domain
ArrayList	Any constructor Any method	java.util
Arrays	Any constructor Any method	java.util
AttributeDef	getAttributeKind() getIndex() getJavaType()	oracle.jbo

	getName() getPrecision() getProperty() getScale() getUIHelper() getUpdateableFlag() isMandatory() isQueriable()	
AttributeHints	getControlType() getDisplayHeight() getDisplayHint() getDisplayWidth() getFormat() getFormattedAttribute() getFormatter() getFormatterClassName() getHint() getLocaleName() parseFormattedAttribute()	oracle.jbo
AttributeList	getAttribute() getAttributeIndexOf() getAttributeNames() setAttribute()	oracle.jbo
BaseLobDomain	closeCharacterStream() closeInputStream() closeOutputStream() getInputStream() getLength() getOutputStream() getcharacterStream()	oracle.jbo.domain
BigDecimal	Any constructor Any method	java.math
BigInteger	Any constructor Any method	java.math

BitSet	Any constructor Any method	java.util
Blob	Any constructor Any method	java.sql
BlobDomain	Any constructor getBinaryOutputStream() getBinaryStream() getBufferSize()	oracle.jbo.domain
Boolean	Any constructor Any method	java.lang
Byte	Any constructor Any method	java.lang
Calendar	Any constructor Any method	java.util
Char	Any constructor bigDecimalValue() bigIntegerValue() booleanValue() doubleValue() floatValue() getValue() intValue() longValue()	oracle.jbo.domain
Clob	Any constructor Any method	java.sql
ClobDomain	Any constructor toCharArray()	oracle.jbo.domain
Collection	Any constructor Any method	java.util
Collections	Any constructor Any method	java.util
Comparator	Any constructor Any method	java.util
Currency	Any constructor Any method	java.util

DBSequence	Any constructor getValue()	oracle.jbo.domain
Date	Any constructor Any method	java.util
Date	Any constructor Any method	java.sql
Date	Any constructor compareTo() dateValue() getValue() stringValue() timeValue() timestampValue()	oracle.jbo.domain
Dictionary	Any constructor Any method	java.util
Double	Any constructor Any method	java.lang
Enum	Any constructor Any method	java.lang
EnumMap	Any constructor Any method	java.util
EnumSet	Any constructor Any method	java.util
Enumeration	Any constructor Any method	java.util
EventListener	Any constructor Any method	java.util
EventListenerProxy	Any constructor Any method	java.util
EventObject	Any constructor Any method	java.util
Exception	Any constructor Any method	java.lang
ExprValueErrorHandler	addAttribute() clearAttributes()	oracle.jbo

	raise() raiseLater() warn()	
Float	Any constructor Any method	java.lang
Formattable	Any constructor Any method	java.util
FormattableFlags	Any constructor Any method	java.util
Formatter	Any constructor Any method	java.util
GregorianCalendar	Any constructor Any method	java.util
HashMap	Any constructor Any method	java.util
HashSet	Any constructor Any method	java.util
Hashtable	Any constructor Any method	java.util
IdentityHashMap	Any constructor Any method	java.util
Integer	Any constructor Any method	java.lang
Iterator	Any constructor Any method	java.util
JboException	getDetails() getErrorCode() getErrorParameters() getLocalizedMessage() getMessage() getProductCode() getProperty()	oracle.jbo
JboWarning	Any constructor getDetails() getErrorCode()	oracle.jbo

	getErrorParameters() getLocalizedMessage() getMessage() getProductCode() getProperty()	
Key	toStringFormat()	oracle.jbo
LinkedHashMap	Any constructor Any method	java.util
LinkedHashSet	Any constructor Any method	java.util
LinkedList	Any constructor Any method	java.util
List	Any constructor Any method	java.util
ListIterator	Any constructor Any method	java.util
ListResourceBundle	Any constructor Any method	java.util
Locale	Any constructor Any method	java.util
Long	Any constructor Any method	java.lang
Map	Any constructor Any method	java.util
Math	Any constructor Any method	java.lang
MathContext	Any constructor Any method	java.math
NClob	Any constructor Any method	java.sql
NameValuePairs	Any constructor getAttribute() getAttributeIndexOf() getAttributeNames() setAttribute()	oracle.jbo

NativeTypeDomainInterface	getNativeObject()	oracle.jbo.domain
Number	Any constructor bigDecimalValue() bigIntegerValue() booleanValue() byteValue() doubleValue() floatValue() getValue() intValue() longValue() shortValue()	oracle.jbo.domain
Observable	Any constructor Any method	java.util
Observer	Any constructor Any method	java.util
PriorityQueue	Any constructor Any method	java.util
Properties	Any constructor Any method	java.util
PropertyPermission	Any constructor Any method	java.util
PropertyResourceBundle	Any constructor Any method	java.util
Queue	Any constructor Any method	java.util
Random	Any constructor Any method	java.util
RandomAccess	Any constructor Any method	java.util
Ref	Any constructor Any method	java.sql
ResourceBundle	Any constructor Any method	java.util
Row	getAttribute()	oracle.jbo

	getAttributeHints() getKey() getLookupDescription() getOriginalAttributeValue() getPrimaryPostState() isAttributeChanged() isAttributeUpdateable() remove() revertRow() revertRowAndContainees() setAttribute() validate()	
RowId	Any constructor Any method	java.sql
RowIterator	createAndInitRow() createRow() findByKey() findRowsMatchingCriteria() first() getAllRowsInRange() getCurrentRow() getEstimatedRowCount() hasNext() hasPrevious() insertRow() last() next() previous() reset()	oracle.jbo
RowSet	avg() count() createAndInitRow() createRow() executeQuery()	oracle.jbo

	findByKey() findRowsMatchingCriteria() first() getAllRowsInRange() getCurrentRow() getEstimatedRowCount() hasNext() hasPrevious() insertRow() last() max() min() next() previous() reset() sum()	
Scanner	Any constructor Any method	java.util
SecurityContext	getUserName() getUserProfile()	oracle.adf.share.security
Session	getLocale() getLocaleContext() getUserData()	oracle.jbo
Set	Any constructor Any method	java.util
Short	Any constructor Any method	java.lang
Short	Any constructor Any method	java.lang
SimpleTimeZone	Any constructor Any method	java.util
SortedMap	Any constructor Any method	java.util
SortedSet	Any constructor Any method	java.util

Stack	Any constructor Any method	java.util
StackTraceElement	Any constructor Any method	java.lang
StrictMath	Any constructor Any method	java.lang
String	Any constructor Any method	java.lang
StringBuffer	Any constructor Any method	java.lang
StringBuilder	Any constructor Any method	java.lang
StringTokenizer	Any constructor Any method	java.util
Struct	Any constructor Any method	java.sql
Struct	getAttribute() setAttribute()	oracle.jbo.domain
StructureDef	findAttributeDef() getAttributeIndexOf()	oracle.jbo
Time	Any constructor Any method	java.sql
TimeZone	Any constructor Any method	java.util
Timer	Any constructor Any method	java.util
TimerTask	Any constructor Any method	java.util
Timestamp	Any constructor Any method	java.sql
Timestamp	Any constructor compareTo() dateValue() getValue() stringValue()	oracle.jbo.domain

	timeValue() timestampValue()	
TreeMap	Any constructor Any method	java.util
TreeSet	Any constructor Any method	java.util
UUID	Any constructor Any method	java.util
UserProfile	getBusinessCity() getBusinessCountry() getBusinessEmail() getBusinessFax() getBusinessMobile() getBusinessPOBox() getBusinessPager() getBusinessPhone() getBusinessPostalAddr() getBusinessPostalCode() getBusinessState() getBusinessStreet() getDateofBirth() getDateofHire() getDefaultGroup() getDepartment() getDepartmentNumber() getDescription() getDisplayName() getEmployeeNumber() getEmployeeType() getFirstName() getGUID() getGivenName() getHomeAddress() getHomePhone()	oracle.adf.share.security.identitymanagment

	getInitials() getJpegPhoto() getLastName() getMaidenName() getManager() getMiddleName() getName() getNameSuffix() getOrganization() getOrganizationalUnit() getPreferredLanguage() getPrinciple() getProperties() getProperty() getTimeZone() getTitle() getUIAccessMode() getUniqueName() getUserID() getUserName() getWirelessAccountNumber()	
ValidationException	getDetails() getErrorCode() getErrorParameters() getLocalizedMessage() getMessage() getProductCode() getProperty()	oracle.jbo
Vector	Any constructor Any method	java.util
ViewCriteria	createAndInitRow() createRow() createViewCriteriaRow() findByKey()	oracle.jbo

	findRowsMatchingCriteria() first() getAllRowsInRange() getCurrentRow() getEstimatedRowCount() hasNext() hasPrevious() insertRow() last() next() previous() reset()	
ViewCriteriaItem	getValue() makeCompound() setOperator() setUpperColumns() setValue()	oracle.jbo
ViewCriteriaItemCompound	ensureItem() getValue() makeCompound() setOperator() setUpperColumns() setValue()	oracle.jbo
ViewCriteriaRow	ensureCriteriaItem() getConjunction() isUpperColumns() setConjunction() setUpperColumns()	oracle.jbo
ViewObject	appendViewCriteria() avg() count() createAndInitRow() createRow() createViewCriteria()	oracle.jbo

	executeQuery() findByKey() findRowsMatchingCriteria() first() getAllRowsInRange() getCurrentRow() getEstimatedRowCount() getMaxFetchSize() hasNext() hasPrevious() insertRow() last() max() min() next() previous() reset() setMaxFetchSize() sum()	
WeakHashMap	Any constructor Any method	java.util

Accessing View Objects in Scripts : Explained

A view object is an Oracle ADF component that simplifies querying and working with business object rows. You access view objects when you write groovy scripts using the expression builder in Oracle Fusion CRM Application Composer. To access view objects, use the newView() function for an object API name. The newView() function accesses a custom or standard view object, and creates a new view object instance that programmatically accesses that business object's rows. For example, a common task you will do with this new view object instance is to query some data. You do this by calling the findByKey() function on the view object to find a row by key.

For more information on groovy scripts, see Oracle Fusion CRM Application Composer Scripting Guide (ID 1354807.1) on My Oracle Support at https://support.oracle.com. In particular, refer to "Accessing the View Object for Programmatic Access to Business Objects" for details on accessing view objects.

This topic:

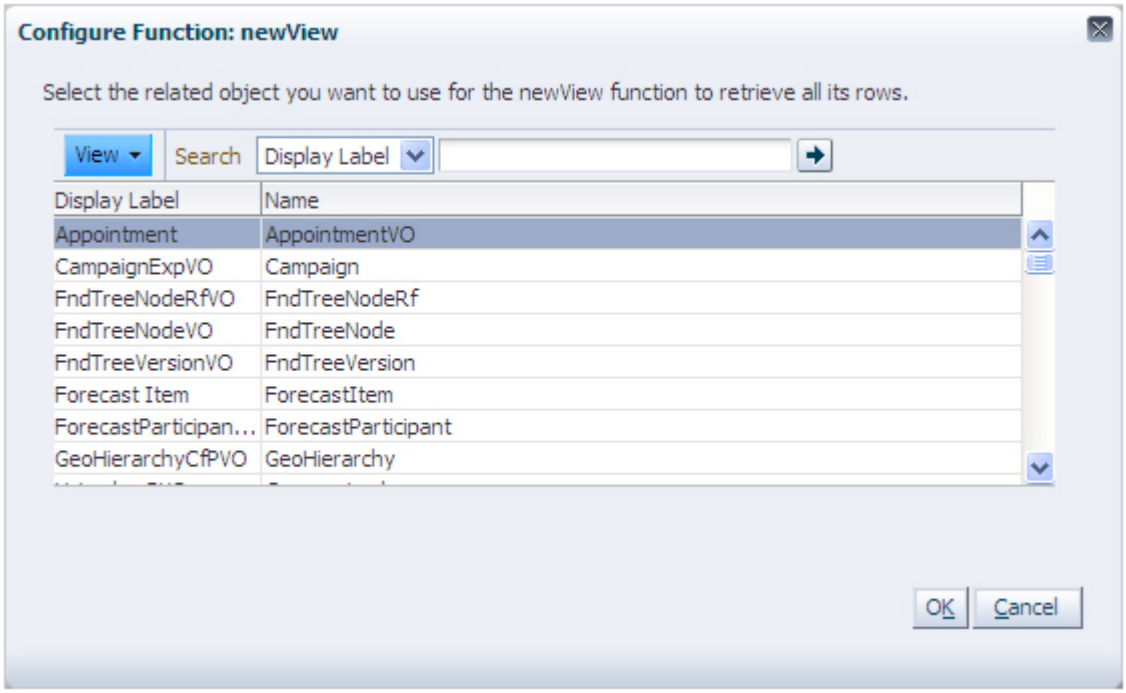
- Explains how to access view objects, either custom or standard, from the expression builder using the newView() function.
- Provides a list of the standard view objects that are provided with the Oracle Fusion Common CRM application.

Accessing View Objects

To access view objects, use the newView() function for an object API name from within the expression builder in Application Composer:

1. Navigate to the expression builder.
2. In the expression builder palette, on the Functions tab, select the Other category and the newView() function.
3. Click Insert.

   A window will display which lists the view objects you can call in your script.



Examples of Standard View Objects

The standard objects that are delivered with Oracle Fusion CRM provide view objects for use in your scripts. The previous section described how to access those view objects. This section provides some examples of standard view objects that are delivered with the Oracle Fusion Common CRM application, and how you might use them in your scripts. Attributes that you would typically script against are also included.

For objects that are not extensible and thus not available in Application Composer, you must refer to Oracle Enterprise Repository for Oracle Fusion Applications to view a list of attributes that you can script against.

1. Go to https://fusionappsoer.oracle.com.

2. Click Login as Guest.
3. Under Type, select ADF Service Data Object.
4. Under Fusion Apps: Logical Business Area, select the desired logical business area, such as Customer Data Management.
5. Click Search.
6. Select the desired object, and click the Details tab to view the attributes you can use in your scripts.

Tip

From OER, you can also review object model diagrams by querying for Data Model Diagram types of records, for a desired logical business area.

Standard View Object	Description	Typical Attributes
Address	Use this object to access the address for a given party in scripting, if the current object does not have a view link to the Address object. Access this Address extensible object as a child of the Organization, Person, or Group object.	Refer to the Trading Community Address object in Application Composer, and review the descriptions provided for all attributes.
CodeAssignment	Use this object to access classifications assigned to a given party in scripting, if the current object does not have a view link to this object. Access this object as a child of the Organization or Person profile objects.	Refer to the Trading Community Classification Code Assignment in OER ADF Service Data Object.
CommonLookup	Access application common lookups in scripting.	LookupType, LookupCode, Tag, EnabledFlag, StartDateActive, EndDateActive, Meaning, Description
Contact	Use this object to access customer contact information in scripting, if the current object does not have a view link to this object. Access this Customer Contact Profile extensible object as a child of the Organization, Person, or Group objects.	Refer to the Trading Community Customer Contact Profile object in Application Composer, and review the descriptions provided for all attributes.
FndTreeVersion	Use this object in scripting to access tree versions. The customer hierarchy and party hierarchy are modeled as trees.	TreeStructureCode, TreeCode, TreeVersionID, Status, EffectiveStartDate, EffectiveEndDate, TreeVersionName
FndTreeNode	Use this object to determine the parent/child relationships for a given hierarchy. The hierarchy for a given version is stored in this object.	TreeStructureCode, TreeCode, TreeVersionID, TreeNodeID, ParentTreeNodeID, Depth, ChildCount, ParentPk1Value
FndTreeNodeRf	Use this object in scripting to easily access the flattened version of the given hierarchy version.	TreeStructureCode, TreeCode, TreeVersionID, TreeNodeID, Pk1Value, AncestorPk1Value, Distance, IsLeaf
Location	Use this object to update or create physical location fields.	Refer to the Trading Community

	Address is the intersection of location and party. Address fields like city, state, and country are stored in the location. These fields are made available in the Address object for read-only purposes. Use this object if you need write access to location fields in scripting.	Location SDO in OER ADF Service Data Object.
OrganizationParty	Use this object to get the organization party and all of its children when you have the organization PartyID in your script, and you do not have a view link from the current object to the Organization Profile object.	Refer to the Trading Community Organization Details in OER ADF Service Data Object.
OrganizationProfile	Access this Organization Profile extensible object as a child of an OrganizationParty row or directly get the profile if you have a PartyID.	Refer to the Trading Community Organization Profile object in Application Composer, and review the descriptions provided for all attributes.
OriginalSystemReference	Use this object to get the ID for given TCA object based on the source system and source system reference information.	Refer to the Trading Community Original System Reference in OER ADF Service Data Object.
PersonParty	Use this object to get the Person Party and all of its children when you have the person PartyID in your script, and you do not have a view link from the current object to Organization Profile object.	Refer to the Trading Community Person Details in OER ADF Service Data Object.
PersonProfile	Access this Person Profile extensible object as a child of a PersonParty row or directly get the profile if you have a PartyID.	Refer to the Trading Community Person Profile object in Application Composer, and review the descriptions provided for all attributes.
Relationship	Use this object in scripting if you have a RelationshipID on the current object and that object does not have a view link to this object. Access this Relationship extensible object as a child of the Organization, Person, or Group object.	Refer to the Trading Community Relationship object in Application Composer, and review the descriptions provided for all attributes.
Resource	Use this Resource extensible object in scripting to get the resource object details if you have a user or resource PartyID, and the current object ID does not expose a view link to this object.	Refer to the Trading Community Resource Profile in OER ADF Service Data Object.

Global Functions : Explained

Global functions are useful for code that multiple objects want to share. You use global functions when you write groovy scripts using the expression builder in Oracle Fusion CRM Application Composer. Some Oracle Fusion CRM applications provide global functions ready for your use, or you can define new global functions.

For more information on global functions, see Oracle Fusion CRM Application Composer Scripting Guide (ID 1354807.1) on My Oracle Support at https://support.oracle.com. In particular, refer to "Defining Utility Code in a Global Function" for details on using global functions in your scripts, including examples.

This topic:

- Explains how to define new global functions.

- Provides a list of some global functions that are provided with the Oracle Fusion Common CRM application.

Defining Global Functions

To define a global function:

1. In Application Composer, navigate to the Common Setup pane, which displays in the regional area.
2. Click Global Functions.
3. On the Global Functions page, click the New icon.
4. When defining a function, specify the global function name and a return value.
5. Optionally specify one or more typed parameters that the caller will be required to pass in when invoked.
6. Specify the body of the function.
7. Validate and save your function.

Examples of Predefined Global Functions

This table lists the global functions that are provided with the Oracle Fusion Common CRM application.

Note

These global functions are not available for selection in the expression builder. Instead, to use these functions, manually type the function name into your script, prefacing the function name with the adf.util. prefix.

Global Function	Description
adf.util.getUserPartyId()	Returns the logged-in user's Party_ID.
adf.util.getUserPartnerCompanyId()	Returns the partner company's party_ID for the logged-in user, if the user is a partner user.
adf.util.getUserRootResourceOrgId()	Returns the organization_ID for the logged-in user's organization hierarchy root resource organization.

Calling Web Services from Groovy Scripts : Explained

You can call Web services from your Groovy scripts in Oracle Fusion CRM Application Composer. You might call a Web service for access to internal or external data, or for example, to perform a calculation on your data.

Calling Web service methods from your scripts involves two high-level steps:

1. Creating a reference to the Web service. This includes registering the Web service with a variable name that you use in your Groovy code.

2. Writing Groovy code in Expression Builder that calls the Web service. For each call, the code must prepare the inbound arguments to the Web service, call a Web service method, and then process the return value from the Web service.

Creating a Web Service Reference

To register a Web service for use in your scripts, you first select **Web Services** in the Common Setup pane in Application Composer. You then associate a Web service variable name with a URL that provides the location of the Web Service Description Language (WSDL) resource that represents the service you want to call.

For example, you might register a Web service variable name of EmployeeService for a Web service that your application needs to invoke for working with employee data from another system. The URL for this Web service's WSDL might be:

```
http://example.com:8099/Services/EmployeeService?WSDL
```

Of course, the server name, the port number, and path name for your actual service will be different. If the port number is omitted, then it is assumed that the service is listening on the default HTTP port number 80.

Read "Creating Web Service References for Groovy Scripts: Explained" for more information about creating Web service references.

Writing Groovy Code to Call a Web Service

When you call a Web service from a Groovy script, the code must prepare the arguments to the Web service before calling a Web service method, and then process the data returned from the Web service. If your code passes structured data to and from a Web service, read "Using Groovy Maps and Lists with Web Services" below.

You insert the code for the call to the Web service from the **Web Services** tab in Expression Builder. As shown in the figure, the **Web Services** list displays the set of registered Web service variable names and the **Functions** list displays the available methods for a given Web service.

To insert a call to a Web service in a Groovy script.

1. Select the **Web Services** tab in Expression Builder.

2. Select a variable name from the **Web Services** list.

3. Select a method from the **Functions** list.

   The code that will be inserted is shown under **Function Signature**.

4. Click the **Insert** button to insert the code to invoke the Web service method.

As you can see in the figure, a Web service call from a Groovy script has the following syntax:

`adf.webServices`.YourServiceVariableName.MethodName(args)

For examples of Groovy code that calls a Web service, read "Web Service Calls in Groovy Scripts: Examples".

Using Groovy Maps and Lists with Web Services

When passing and receiving structured data to and from a Web service, a Groovy map represents an object and its properties. For example, an Employee object with properties named Empno, Ename, Sal, and Hiredate would be represented by a map object having four key-value pairs, where the names of the properties are the keys.

You can create an empty Map object using the syntax:

```
def newEmp = [:]
```

Then, you can add properties to the map using the explicit put() method like this:

```
newEmp.put("Empno",1234)
newEmp.put("Ename","Sean")
newEmp.put("Sal",9876)
newEmp.put("Hiredate",date(2013,8,11))
```

Alternatively, and more conveniently, you can assign and update map key-value pairs using a simpler direct assignment notation like this:

```
newEmp.Empno = 1234
newEmp.Ename = "Sean"
newEmp.Sal = 9876
newEmp.Hiredate = date(2013,8,11)
```

Finally, you can also create a new map and assign some or all of its properties in a single operation using the constructor syntax:

```
def newEmp = [Empno : 1234,
              Ename : "Sean",
              Sal : 9876,
              Hiredate : date(2013,8,11)]
```

To create a collection of objects you use the Groovy List object. You can create one object at a time and then create an empty list, and call the list's add() method to add both objects to the list:

```
def dependent1 = [Name : "Dave",
                  BirthYear : 1996]
def dependent2 = [Name : "Jenna",
                  BirthYear : 1999]
def listOfDependents = []
listOfDependents.add(dependent1)
listOfDependents.add(dependent2)
```

To save a few steps, the last three lines in the preceding example can be done in a single line by constructing a new list with the two desired elements in one line like this:

```
def listOfDependents = [dependent1, dependent2]
```

You can also create the list of maps in a single operation using a combination of list constructor syntax and map constructor syntax:

```
def listOfDependents = [[Name : "Dave",
                         BirthYear : 1996],
                        [Name : "Jenna",
                         BirthYear : 1999]]
```

If the employee object in the previous codes examples has a property named Dependents that is a list of objects representing dependent children, you can assign the property using the same syntax as shown above (using a list of maps as the value assigned):

```
newEmp.Dependents = [[Name : "Dave",
                      BirthYear : 1996],
                     [Name : "Jenna",
                      BirthYear : 1999]]
```

Lastly, note that you can also construct a new employee with nested dependents all in a single statement by further nesting the constructor syntax:

```
def newEmp = [Empno : 1234,
              Ename : "Sean",
              Sal : 9876,
              Hiredate : date(2013,8,11),
              Dependents : [
                   [Name : "Dave",
                    BirthYear : 1996],
                   [Name : "Jenna",
                    BirthYear : 1999]]
              ]
```

For more information on maps and lists, see the section called Working with Maps in the Oracle Fusion CRM Application Composer Scripting Guide (ID 1354807.1) on My Oracle Support at https://support.oracle.com.

Web Service References for Groovy Scripts : Explained

In the Groovy scripts that you use in Oracle Fusion CRM Application Composer, you can include calls to both internal and external Web services. For each Web service that you call in your scripts, you must set up a Web service reference that specifies the Web Services Description Language (WSDL) file location and the security scheme, if any, used to access the Web service.

To create a Web service reference, do the following in Application Composer:

1. Select **Web Services** in the Common Setup pane.
2. On the Web Services page, click the **New** icon.
3. Specify a name for the Web service connection.
4. Specify the URL of the WSDL file for the Web service.

5. Specify the user and password credentials as required for the security scheme for the Web service. Read "Specifying the Security Values for the Web Service" below for information about which schemes are supported.

After you create a Web service reference, the name of the Web service appears in the list available in the Web services tab in the Expression Builder. When you select a Web service from the list, you can then select any of the functions provided by the Web service for use in your Groovy scripts.

Specifying Variable Names

When you create a Web service reference, you specify a variable name on the Create SOAP Web Service Connection page. This name is simply an identifier that is used in the list of Web services in the Expression Builder.

Specifying WSDL URLs

The WSDL file for a Web service provides information about a Web service that includes the following:

- **Service.** Defines one or more ports for the Web service.
- **Port.** Defines an address or connection endpoint to the Web service.

For each service and port there can be one or more associated security policies.

To specify a WSDL URL:

1. On the Create SOAP Web Service Connection page, enter the WSDL file in URL format, for example:

   ```
   http://internal-hosted:7101/MathsWS-Model-context-root/UsernameTokenSecurity?wsdl
   ```

2. Click **Read WSDL**.

   The **Service**, **Port**, and **Security Scheme** fields are then populated based on what is found in the WSDL. When there are multiple services and ports defined, the **Service** and **Port** dropdowns have the first service and port found in the WSDL selected.

3. If a different service and port is required for this Web service, select the appropriate values in **Service** and **Port**.

   When you select a particular service and port, a default security scheme is selected based on the security policy defined in the WSDL.

   If the port number is omitted, then it is assumed that the service is listening on the default HTTP port number 80.

Specifying the Security Values for the Web Service

For secure communication with a Web service, you can use various schemes for authenticating user credentials and ensuring security. The following schemes are supported for Web services from Groovy scripts:

- Using separate user name and password credentials over Secure Sockets Layer (SSL)

- Using separate user name and password credentials with message protection
- Using single sign-on through Security Assertion Markup Language (SAML).

You can also specify that no security scheme is used.

On the Create SOAP Web Service Connection page you specify a credential key for the security schemes that require user name and password credentials. The Web service provider will tell you about the credentials that you must use for a particular Web service.

Resolving Security Setup Errors

You may receive some errors if some security setup has not been performed. For example, you may get a SSL certificate error when you try to create the Web service reference. In this case, you must create a service request for your administrator. You must retrieve the server's CA SSL certificate from the service provider and attach it in the service request along with the WSDL location, and error details. The administrator will import the server SSL certificate into the tenant domain and inform you when this has happened.

You may also receive errors when the Web service is called from a Groovy script:

- A bad encryption error, when message protection is used
- A PolicyEnforcementException error when message protection security is used.

For these errors you must also create a service request for your administrator to resolve the errors. You must retrieve the server's encryption certificate and the issuer certificate from the service provider and attach them both in the service request along with the WSDL location and the error details.

Using Worked Examples of Calling Web Services from Groovy

Worked examples of creating Web service connections and calling the Web service from a Groovy script are provided in separate topics as listed under "Related Links" below.

The topics cover the various security schemes that are supported for calls to both internal and external Web services. The topics include information about contacting your administrator to resolve security setup errors where appropriate.

Web Service Calls in Groovy Scripts : Examples

You can call Web services from your Groovy scripts in Oracle Fusion CRM Application Composer, for example, to access internal or external data, or to perform a calculation on your data. This topic provides examples of calling Web service methods from Groovy scripts.

A Web service call from a Groovy script has the following syntax:

`adf.webServices`.YourServiceVariableName.MethodName(args)

In the examples in this topic, methods of a Web service registered with the variable name EmployeeService are called.

Note

For each Web service that you call in your scripts, you must set up a Web service reference in the Web Services page in Oracle Fusion CRM Application Composer.

Retrieving an Employee by Id

The following example shows how to call a getEmployee() method of the Web service, passing the integer 7839 as the single argument to the method.

```
// retrieve Employee object by id from remote system
def emp = adf.webServices.EmployeeService.getEmployee(7839)
// log a message, referencing employee fields with "dot" notation
println('Got employee '+emp.Ename+' with id '+emp.Empno)
// access the nested list of Dependent objects for this employee
def deps = emp.Dependents
if (deps != null) {
  println("Found "+deps.size()+" dependents")
  for (dep in deps) {
    println("Dependent:"+dep.Name)
  }
}
```

Creating a New Employee Including New Dependents

The following example shows how to use Groovy's convenient map and list construction notation to create a new employee with two nested dependents. The newEmp object is then passed as the argument to the createEmployee() method of the Web service.

```
// Create a new employee object using a Groovy map. The
// nested collection of dependents is a Groovy list of maps
def newEmp = [ Ename:"Steve",
               Deptno:10,
                  Job:"CLERK",
                  Sal:1234,
               Dependents:[[Name:"Timmy",BirthYear:1996],
                           [Name:"Sally",BirthYear:1998]]]
// Create the new employee by passing this object to a web service
newEmp = adf.webServices.EmployeeService.createEmployee(newEmp)
// The service returns a new employee object which may have
// other attributes defaulted/assigned by the service, like the Empno
println("New employee created was assigned Empno = "+ newEmp.Empno)
```

Merging Updates to an Employee Object and Adding a New Child Dependent Object

The following example shows how to use the mergeEmployee() method to update fields in an employee object that is retrieved at the start of the script using a call to the getEmployee() method. The script updates the Ename field on the emp object retrieved, updates the names of the existing dependents, and then adds a new dependent before calling the mergeEmployee() method of the Web service to save the changes.

```
// Merge updates and inserts on Employee and nested Dependents
def emp = adf.webServices.EmployeeService.getEmployee(7839)
```

```
// update employee's name to add an exclamation point!
emp.Ename = emp.Ename + '!'
def deps = emp.Dependents
// Update dependent names to add an exclamation point!
for (dep in deps) {
   dep.Name = dep.Name + '!'
}
// Add a new dependent
def newChild = [Name:"Jane", BirthYear:1997]
deps.add(newChild)
emp = adf.webServices.EmployeeService.mergeEmployee(emp)
```

Calling an External Web Service from Groovy when No Security Scheme is Required : Worked Example

This example shows how to create a connection to an external Web service on the Internet and call the Web service from a Groovy script used in Oracle Fusion CRM Application Composer. The Web service is not secured. For this example, the Web service is used to calculate a custom field's default value.

The following table summarizes key decisions for this scenario:

Decisions to Consider	In This Example
What name will you use for the Web service connection?	mathsws
What is the URL of the Web Services Description Language (WSDL) file that you will use?	http://external-hosted:7101/MathsWS-Model-context-root/NoSecurity?wsdl **Note** The URL shown here is an arbitrary example. You must obtain the real WSDL URL from the service provider,
Where will the Web service be called from?	From a Groovy script expression used to calculate a custom field's default value.
Which Web service method will be called from the Groovy script?	getSum This method returns the sum of two integer argument values.

To call a Web service from a Groovy script when no security scheme is required, complete the following tasks:

1. Create the Web service connection.
2. Add the Web service call to the Groovy script, and verify that the call succeeds.

Prerequisites

Verify that you have completed the following prerequisite steps:

1. Get details of the WSDL URL to use from the Web service provider.
2. Create a custom field for an object that has a calculated default value.
3. Prepare the Groovy script for the expression used to calculate the field's default value. The Groovy code must prepare the argument values, which in this example are two values that are summed.

Creating the Web Service Connection

When you create a Web service connection, you specify a name for the Web service, the URL of the WSDL file, and the security scheme settings. The name is simply an identifier that is used in the list of Web services in the Expression Builder in Application Composer.

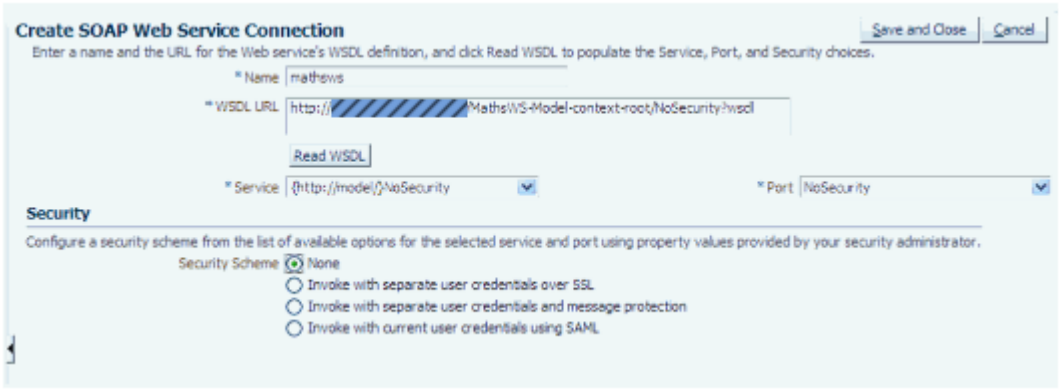
1. In Application Composer, select **Web Services** in the Common Setup pane.
2. On the Web Services page, click the New icon.
3. On the Create SOAP Web Service Connection page, enter `mathsws` in the **Name** field.

   The name must not include periods.

4. Enter `http://external-hosted:7101/MathsWS-Model-context-root/NoSecurity?wsdl` in the **WSDL URL** field, and click **Read WSDL**.

   After you click **Read WSDL**. the **Service** and **Port** fields are filled according to values in the WSDL file. Under **Security Scheme**, the **None** radio button becomes enabled and selected.

   This figure shows the Create SOAP Web Service Connection page.



5. Click **Save and Close**.

   The Web service connection is created and the name and WSDL URL are listed on the Web Services page.

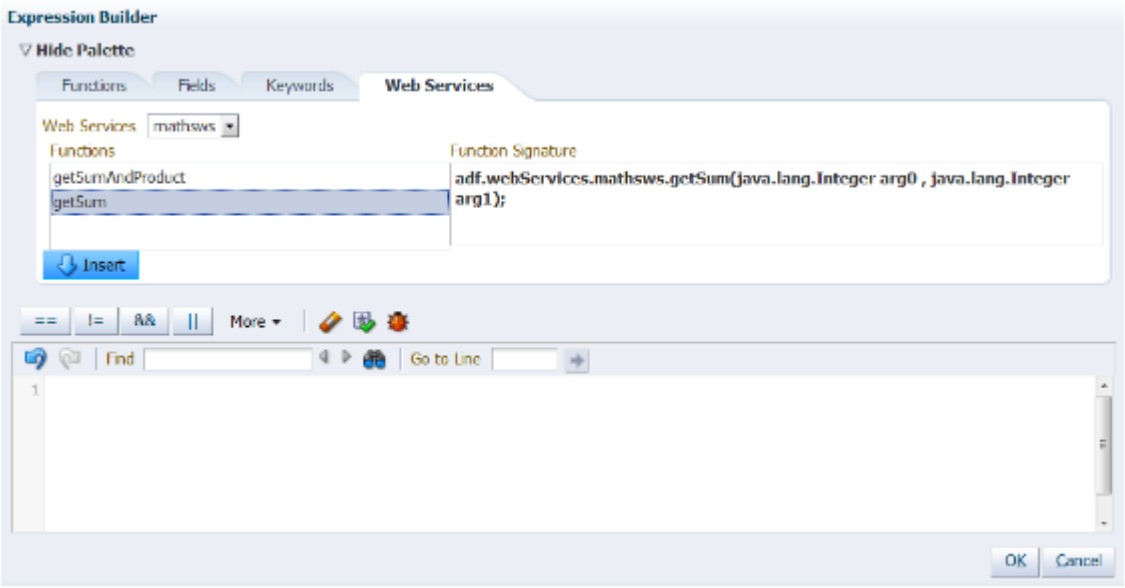
Adding the Web Service Call to the Groovy Script

In the Expression Builder dialog that you see when you create or edit Groovy scripts, there is a **Web Services** tab that lists the Web services for which you have created a connection. For each Web service you can include calls to the available methods in your Groovy script.

1. In Application Composer, edit the custom field that uses the expression that will contain the Web service call.
2. Click the Expression Builder icon.
3. In the Expression Builder dialog, select the **Web Services** tab.

4. Select `mathsws` from the **Web Services** list.
5. Select **getSum** from the **Functions** list.

   The code that will be inserted is shown under **Function Signature**, as illustrated in the figure.



6. Position the cursor at the place in the script where you want to insert the Web service call.
7. Click **Insert** to insert the code to invoke the Web service method.
8. Update the script so that two integer values are provided as arguments for the Web service call.
9. Click **Submit**.
10. Verify that the Web service call succeeds; in this example the custom field should have the expected default value.

Calling an External Web Service from Groovy with Message Protection : Worked Example

This example shows how to create a connection to an external, secured Web service and call the Web service from a Groovy script used in Oracle Fusion CRM Application Composer. The Web service is secured with message protection. For this example, the Web service is used to calculate a custom field's default value.

The following table summarizes key decisions for this scenario:

Decisions to Consider	In This Example
What name will you use for the Web service connection?	mathsws
What is the URL of the Web Services Description Language (WSDL) file that you will use?	http://external-hosted:7101/MathsWS-Model-context-root/Wss11UsernameWithMessageProtectionSecurity?wsdl This WSDL file specifies the desired message protection security scheme. **Note**

	The URL shown here is an arbitrary example. You must obtain the real WSDL URL from the service provider,
Which credential key will you use?	mylogin
Where will the Web service be called from?	From a Groovy script expression used to calculate a custom field's default value.
Which Web service method will be called from the Groovy script?	getSum This method returns the sum of two integer argument values.
What will the server encryption alias name be?	serverenckey

To call a Web service from a Groovy script that is secured with message protection, complete the following tasks:

1. Create the Web service connection.
2. Add the Web service call to the Groovy script, and check whether the call succeeds.
3. Contact the administrator to resolve runtime exceptions
4. Re-create the Web service connection.
5. Verify that the Web service call succeeds.

Prerequisites

Verify that you have completed the following prerequisite steps:

1. Get details of the WSDL URL and the user credentials to use from the Web service provider.
2. Get the server encryption certificate and the Certificate Authority (issuer) certificate from the Web service provider.
3. Create a custom field for an object that has a calculated default value.
4. Prepare the Groovy script for the expression used to calculate the field's default value. The Groovy code must prepare the argument values, which in this example are two values that are summed.

Creating the Web Service Connection

When you create a Web service connection, you specify a name for the Web service, the URL of the WSDL file, and the security scheme settings. The name is simply an identifier that is used in the list of Web services in the Expression Builder in Application Composer.

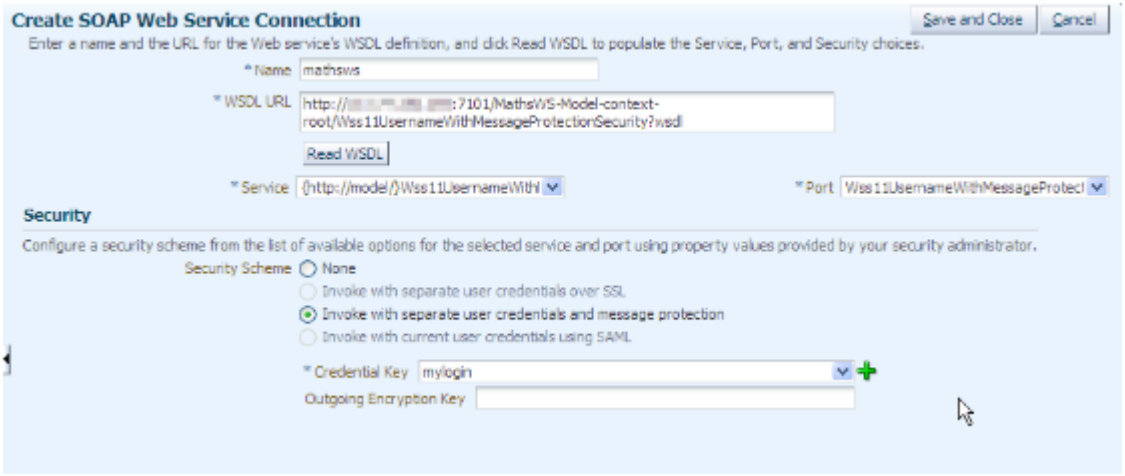
1. In Application Composer, select **Web Services** in the Common Setup pane.
2. On the Web Services page, click the New icon.
3. On the Create SOAP Web Service Connection page, enter `mathsws` in the **Name** field.

   The name must not include periods.

4. Enter `http://external-hosted:7101/MathsWS-Model-context-root/Wss11UsernameWithMessageProtectionSecurity?wsdl` in the **WSDL URL** field, and click **Read**

WSDL.

The following figure shows what happens after you click **Read WSDL**. The **Service** and **Port** fields are filled according to values in the WSDL file. Under **Security Scheme**, the **Invoke with separate user credentials and message protection** radio button becomes enabled and selected and the **Credential Key** and **Outgoing Encryption Key**fields appear.



5. Click the New Key icon next to the **Credential Key** field.
6. In the Create Key dialog box, enter a name in the **Credential Key** field, in this example, `mylogin`, enter the user name and password credentials supplied by the Web service provider, and click OK.
7. Click **Save and Close**.

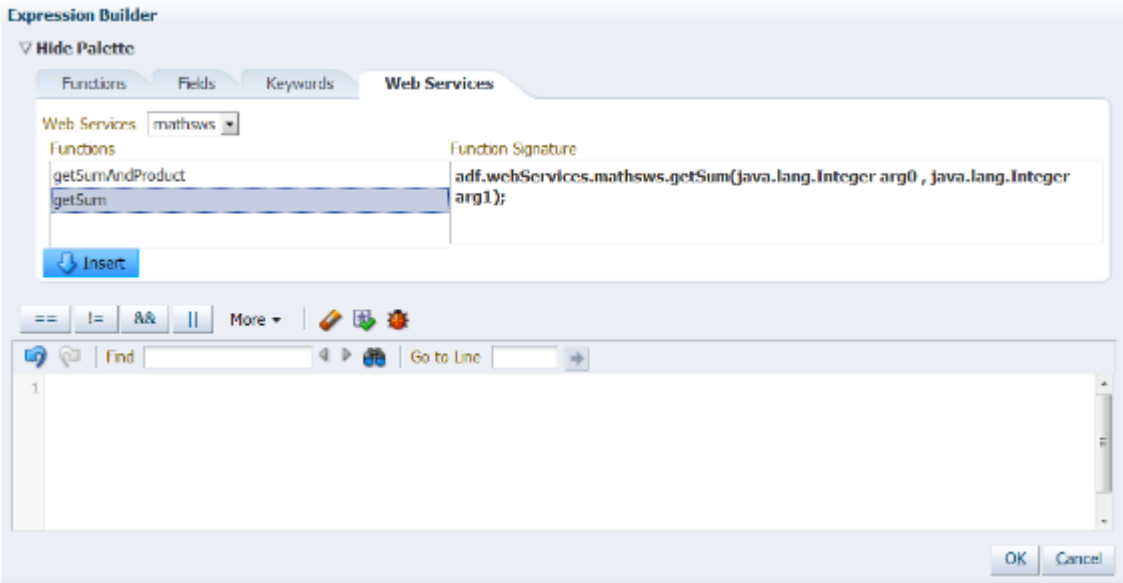
   The Web service connection is created and the name and WSDL URL are listed on the Web Services page.

Adding the Web Service Call to the Groovy Script

In the Expression Builder dialog that you see when you create or edit Groovy scripts, there is a **Web Services** tab that lists the Web services for which you have created a connection. For each Web service you can include calls to the available methods in your Groovy script.

1. In Application Composer, edit the custom field that uses the expression that will contain the Web service call.
2. Click the Expression Builder icon.
3. In the Expression Builder dialog, select the **Web Services** tab.
4. Select `mathsws` from the **Web Services** list.
5. Select **getSum** from the **Functions** list.

   The code that will be inserted is shown under **Function Signature**, as illustrated in the figure.



6. Position the cursor at the place in the script where you want to insert the Web service call.
7. Click **Insert** to insert the code to invoke the Web service method.
8. Update the script so that two integer values are provided as arguments for the Web service call.
9. Click **Submit**.
10. Verify that the Web service call succeeds; in this example the custom field should have the expected default value.

Contacting the Administrator to Resolve Runtime Exceptions

The Web service call may fail due to a number of exceptions including path certification, bad encryption, and policy enforcement exceptions. You must create a service request for your administrator to resolve the issues.

1. Create a service request for your administrator:

    a. Retrieve the server encryption certificate and the Certificate Authority (issuer) certificate from the Web service provider.

    b. Attach the server encryption certificate and the issuer certificate to the service request, and include the WSDL location, and error details

    c. Submit the service request.

    The administrator will add the server encryption certificate and the issuer certificate into the Oracle Fusion CRM trust store. The administrator also creates an alias for the server encryption key, which you will use in the next task.

2. Wait until your administrator informs you that the certificates have been imported, and that the server encryption alias has been created, and then close the service request.

Re-creating the Web Service Connection

After your administrator has resolved runtime exceptions, you must re-create the Web service connection and this time specify the server encryption key alias supplied by the administrator.

1. In Application Composer, select **Web Services** in the Common Setup pane.

2. On the Web Services page, select the Web service connection you created previously, and click the Delete icon.
3. On the Web Services page, click the New icon.
4. On the Create SOAP Web Service Connection page, enter `mathsws` in the **Name** field.
5. Enter `http://external-hosted:7101/MathsWS-Model-context-root/Wss11UsernameWithMessageProtectionSecurity?wsdl` in the **WSDL URL** field, and click **Read WSDL**.
6. Click the New Key icon next to the **Credential Key** field.
7. In the Create Key dialog box, enter a name in the **Credential Key** field, in this example, `mylogin`, enter the user name and password credentials supplied by the Web service provider, and click OK.
8. On the Create SOAP Web Service Connection page, enter `serverenckey` in the **Outgoing Encryption Key** field.
9. Click **Save and Close**.

   The Web service connection is created and the name and WSDL URL are listed on the Web Services page.

Verifying that the Web Service Call Succeeds

After you have re-created a Web service connection, you must verify that the call to the Web service succeeds.

1. Make sure that the Groovy script contains the code to call the Web service.
2. Verify that the Web service call succeeds; in this example the custom field should have the expected default value.

Calling an External Web Service from Groovy with Separate User Credentials over SSL : Worked Example

This example shows how to create a connection to an external, secured Web service and call the Web service from a Groovy script used in Oracle Fusion CRM Application Composer. The Web service uses a security scheme with separate user credentials and secure sockets layer (SSL). For this example, the Web service is used to calculate a custom field's default value.

The following table summarizes key decisions for this scenario:

Decisions to Consider	**In This Example**
What name will you use for the Web service connection?	mathsws
What is the URL of the Web Services Description Language (WSDL) file that you will use?	https://external-hosted:7102/MathsWS-Model-context-root/UsernameTokenOverSSLSecurity?wsdl This WSDL file specifies the desired SSL security scheme. **Note** The URL shown here is an arbitrary example. You must obtain the real WSDL URL from the service provider,

Which credential key will you use?	mylogin
Where will the Web service be called from?	From a Groovy script expression used to calculate a custom field's default value.
Which Web service method will be called from the Groovy script?	getSum This method returns the sum of two integer argument values.

To call a Web service from a Groovy script that is secured with SSL, complete the following tasks:

1. Create the Web service connection.
2. Add the Web service call to the Groovy script, and verify that the call succeeds.

Prerequisites

Verify that you have completed the following prerequisite steps:

1. Get details of the WSDL URL and the user credentials to use from the Web service provider.
2. Get the server's Certificate Authority (CA) SSL certificate from the Web service provider
3. Create a custom field for an object that has a calculated default value.
4. Prepare the Groovy script for the expression used to calculate the field's default value. The Groovy code must prepare the argument values, which in this example are two values that are summed.

Creating the Web Service Connection

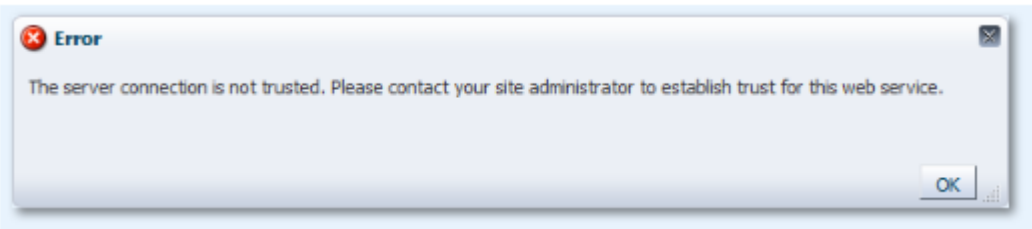
When you create a Web service connection, you specify a name for the Web service, the URL of the WSDL file, and the security scheme settings. The name is simply an identifier that is used in the list of Web services in the Expression Builder in Application Composer.

1. In Application Composer, select **Web Services** in the Common Setup pane.
2. On the Web Services page, click the New icon.
3. On the Create SOAP Web Service Connection page, enter `mathsws` in the **Name** field.

   The name must not include periods.

4. Enter `https://external-hosted:7102/MathsWS-Model-context-root/UsernameTokenOverSSLSecurity?wsdl` in the **WSDL URL** field, and click **Read WSDL**.

   The following figure shows the error that is displayed after you click **Read WSDL**.



   You must create a service request for your administrator to resolve the issue.

5. Create a service request for your administrator:

    a. Retrieve the server's Certificate Authority (CA) SSL certificate from the Web service provider.

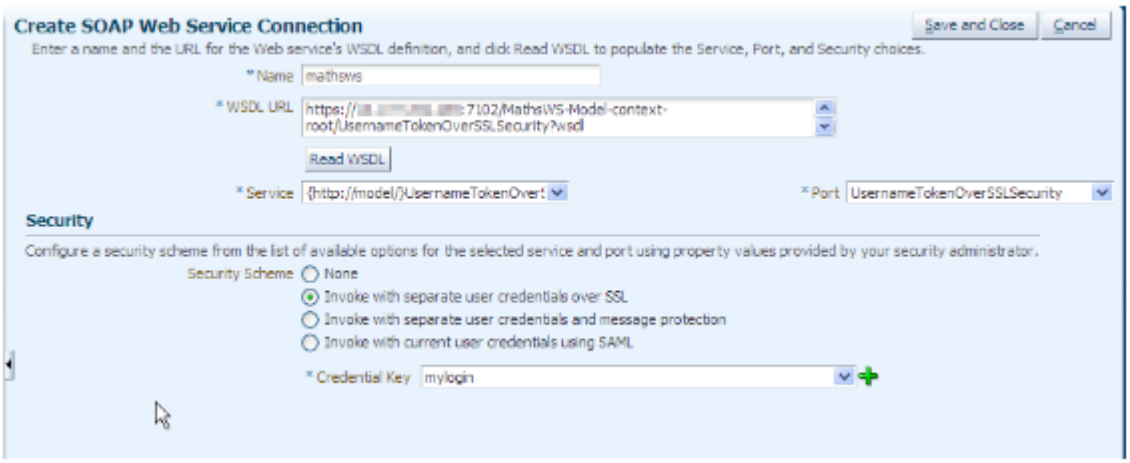
    b. Attach the SSL certificate to the service request, and include the WSDL location, and error details

    c. Submit the service request.

    The administrator will add the SSL certificate into the Oracle Fusion CRM trust store.

6. Wait until your administrator informs you that the SSL certificate has been imported, and close the service request.
7. Repeat steps 1 through 4.

    The following figure shows what happens after you click **Read WSDL**. The **Service** and **Port** fields are filled according to values in the WSDL file. Under **Security Scheme**, the **Invoke with separate user credentials over SSL** radio button becomes enabled and selected and the **Credential Key** field appears.



8. Click the New Key icon next to the **Credential Key** field.
9. In the Create Key dialog box, enter a name in the **Credential Key** field, in this example, `mylogin`, enter the user name and password credentials supplied by the Web service provider, and click OK.
10. Click **Save and Close**.

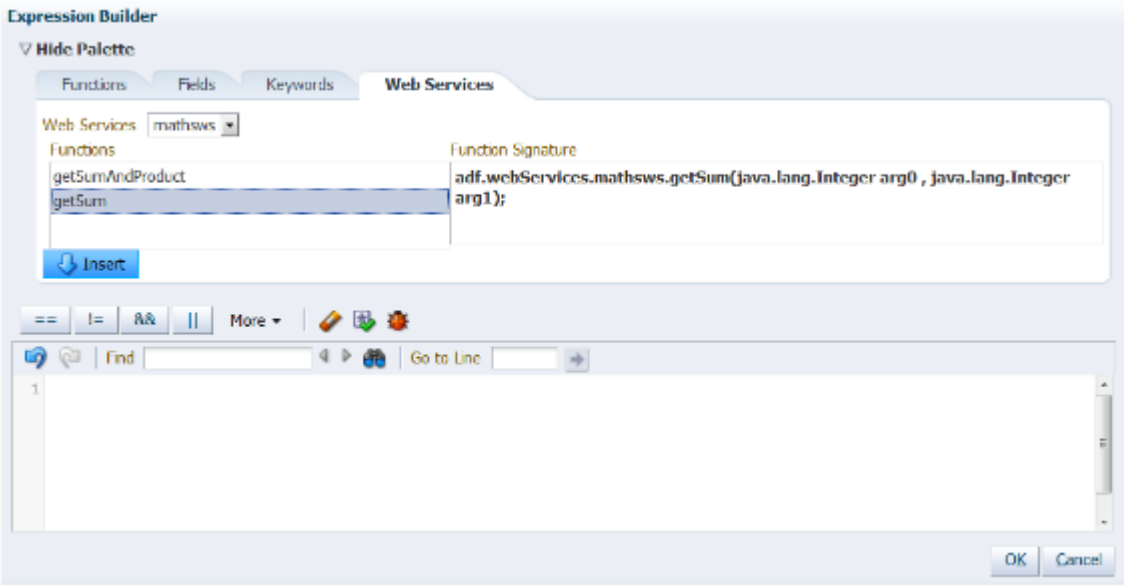
    The Web service connection is created and the name and WSDL URL are listed on the Web Services page.

Adding the Web Service Call to the Groovy Script

In the Expression Builder dialog that you see when you create or edit Groovy scripts, there is a **Web Services** tab that lists the Web services for which you have created a connection. For each Web service you can include calls to the available methods in your Groovy script.

1. In Application Composer, edit the custom field that uses the expression that will contain the Web service call.
2. Click the Expression Builder icon.
3. In the Expression Builder dialog, select the **Web Services** tab.
4. Select `mathsws` from the **Web Services** list.
5. Select **getSum** from the **Functions** list.

The code that will be inserted is shown under **Function Signature**, as illustrated in the figure.



6. Position the cursor at the place in the script where you want to insert the Web service call.
7. Click **Insert** to insert the code to invoke the Web service method.
8. Update the script so that two integer values are provided as arguments for the Web service call.
9. Click **Submit**.
10. Verify that the Web service call succeeds; in this example the custom field should have the expected default value.

Calling an Internal Web Service from Groovy with Separate User Credentials over SSL : Worked Example

This example shows how to create a connection to a Fusion Applications Web service and call the Web service from a Groovy script used in Oracle Fusion CRM Application Composer. The Web service uses a security scheme with separate user credentials and secure sockets layer (SSL). For this example, the Web service is used to calculate a custom field's default value.

The following table summarizes key decisions for this scenario:

Decisions to Consider	In This Example
What name will you use for the Web service connection?	mathsws
What is the URL of the Web Services Description Language (WSDL) file that you will use?	https://internal-hosted:7102/MathsWS-Model-context-root/UsernameTokenOverSSLSecurity?wsdl This WSDL file specifies the desired SSL authentication scheme. **Note** The URL shown here is an arbitrary example. You must obtain the real WSDL URL from the service provider,

Which credential key will you use?	mylogin
Where will the Web service be called from?	From a Groovy script expression used to calculate a custom field's default value.
Which Web service method will be called from the Groovy script?	getSum This method returns the sum of two integer argument values.

To call a Web service from a Groovy script that is secured with SSL, complete the following tasks:

1. Create the Web service connection.
2. Add the Web service call to the Groovy script, and verify that the call succeeds.

Prerequisites

Verify that you have completed the following prerequisite steps:

1. Get details of the WSDL URL and the user credentials to use from the Web service provider.
2. Create a custom field for an object that has a calculated default value.
3. Prepare the Groovy script for the expression used to calculate the field's default value. The Groovy code must prepare the argument values, which in this example are two values that are summed.

Creating the Web Service Connection

When you create a Web service connection, you specify a name for the Web service, the URL of the WSDL file, and the security scheme settings. The name is simply an identifier that is used in the list of Web services in the Expression Builder in Application Composer.

1. In Application Composer, select **Web Services** in the Common Setup pane.
2. On the Web Services page, click the New icon.
3. On the Create SOAP Web Service Connection page, enter `mathsws` in the **Name** field.

   The name must not include periods.

4. Enter `https://internal-hosted:7102/MathsWS-Model-context-root/UsernameTokenOverSSLSecurity?wsdl` in the **WSDL URL** field, and click **Read WSDL**.

   The following figure shows what happens after you click **Read WSDL**. The **Service** and **Port** fields are filled according to values in the WSDL file. Under **Security Scheme**, the **Invoke with separate user credentials over SSL** radio button becomes enabled and selected and the **Credential Key** field appears.

5. Click the New Key icon next to the **Credential Key** field.
6. In the Create Key dialog box, enter a name in the **Credential Key** field, in this example, `mylogin`, enter the user name and password credentials supplied by the Web service provider, and click OK.
7. Click **Save and Close**.

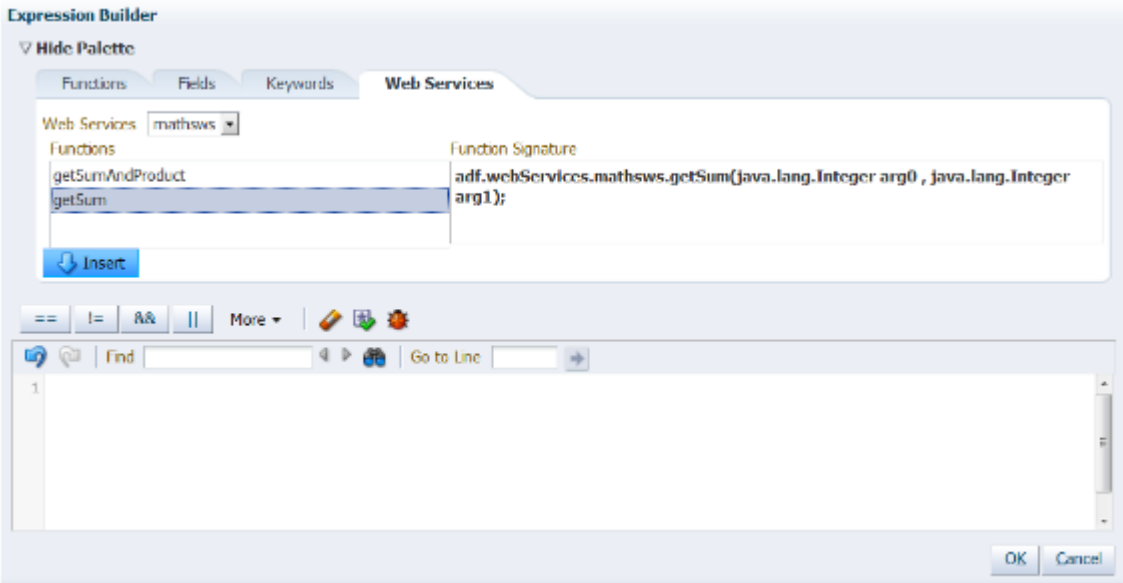
   The Web service connection is created and the name and WSDL URL are listed on the Web Services page.

Adding the Web Service Call to the Groovy Script

In the Expression Builder dialog that you see when you create or edit Groovy scripts, there is a **Web Services** tab that lists the Web services for which you have created a connection. For each Web service you can include calls to the available methods in your Groovy script.

1. In Application Composer, edit the custom field that uses the expression that will contain the Web service call.
2. Click the Expression Builder icon.
3. In the Expression Builder dialog, select the **Web Services** tab.
4. Select `mathsws` from the **Web Services** list.
5. Select **getSum** from the **Functions** list.

   The code that will be inserted is shown under **Function Signature**, as illustrated in the figure.



6. Position the cursor at the place in the script where you want to insert the Web service call.
7. Click **Insert** to insert the code to invoke the Web service method.
8. Update the script so that two integer values are provided as arguments for the Web service call.
9. Click **Submit**.
10. Verify that the Web service call succeeds; in this example the custom field should have the expected default value.

Calling an Internal Web Service with Message Protection Security : Worked Example

This example shows how to create a connection to a Fusion Applications Web service and call the Web service from a Groovy script used in Oracle Fusion CRM Application Composer. The Web service is secured with message protection. For this example, the Web service is used to calculate a custom field's default value.

The following table summarizes key decisions for this scenario:

Decisions to Consider	In This Example
What name will you use for the Web service connection?	mathsws
What is the URL of the Web Services Description Language (WSDL) file that you will use?	http://internal-hosted:7101/MathsWS-Model-context-root/Wss11UsernameWithMessageProtectionSecurity?wsdl This WSDL file specifies the desired message protection security scheme. **Note** The URL shown here is an arbitrary example. You must obtain the real WSDL URL from the service provider,

Which credential key will you use?	mylogin
Where will the Web service be called from?	From a Groovy script expression used to calculate a custom field's default value.
Which Web service method will be called from the Groovy script?	getSum This method returns the sum of two integer argument values.

To call a Web service from a Groovy script that is secured with message protection, complete the following tasks:

1. Create the Web service connection.
2. Add the Web service call to the Groovy script, and verify that the call succeeds.

Prerequisites

Verify that you have completed the following prerequisite steps:

1. Get details of the WSDL URL and the user credentials to use from the Web service provider.
2. Create a custom field for an object that has a calculated default value.
3. Prepare the Groovy script for the expression used to calculate the field's default value. The Groovy code must prepare the argument values, which in this example are two values that are summed.

Creating the Web Service Connection

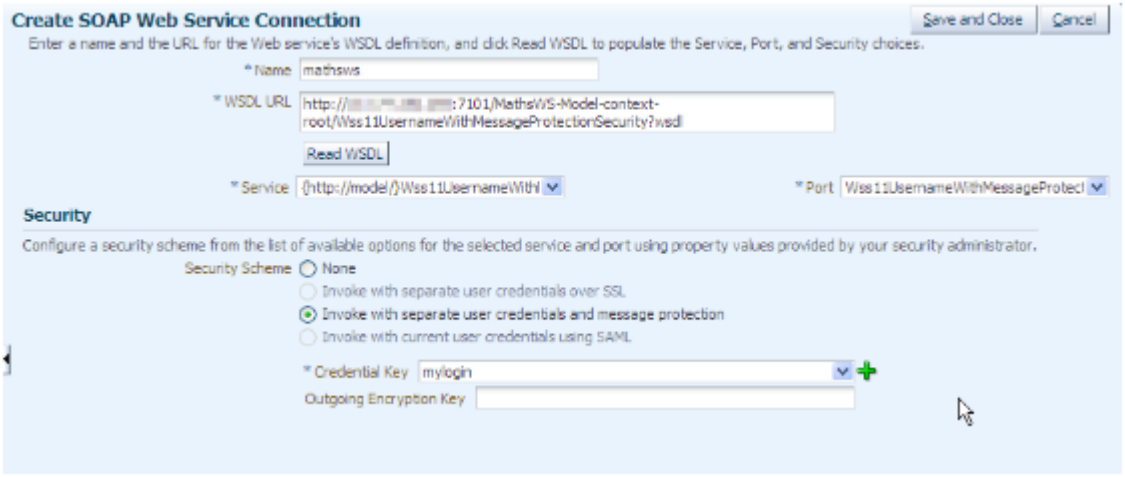
When you create a Web service connection, you specify a name for the Web service, the URL of the WSDL file, and the security scheme settings. The name is simply an identifier that is used in the list of Web services in the Expression Builder in Application Composer.

1. In Application Composer, select **Web Services** in the Common Setup pane.
2. On the Web Services page, click the New icon.
3. On the Create SOAP Web Service Connection page, enter `mathsws` in the **Name** field.

   The name must not include periods.

4. Enter `http://internal-hosted:7101/MathsWS-Model-context-root/Wss11UsernameWithMessageProtectionSecurity?wsdl` in the **WSDL URL** field, and click **Read WSDL**.

   The following figure shows what happens after you click **Read WSDL**. The **Service** and **Port** fields are filled according to values in the WSDL file. Under **Security Scheme**, the **Invoke with separate user credentials and message protection** radio button becomes enabled and selected and the **Credential Key** and **Outgoing Encryption Key**fields appear.



5. Click the New Key icon next to the **Credential Key** field.
6. In the Create Key dialog box, enter a name in the **Credential Key** field, in this example, `mylogin`, enter the user name and password credentials supplied by the Web service provider, and click OK.
7. Click **Save and Close**.

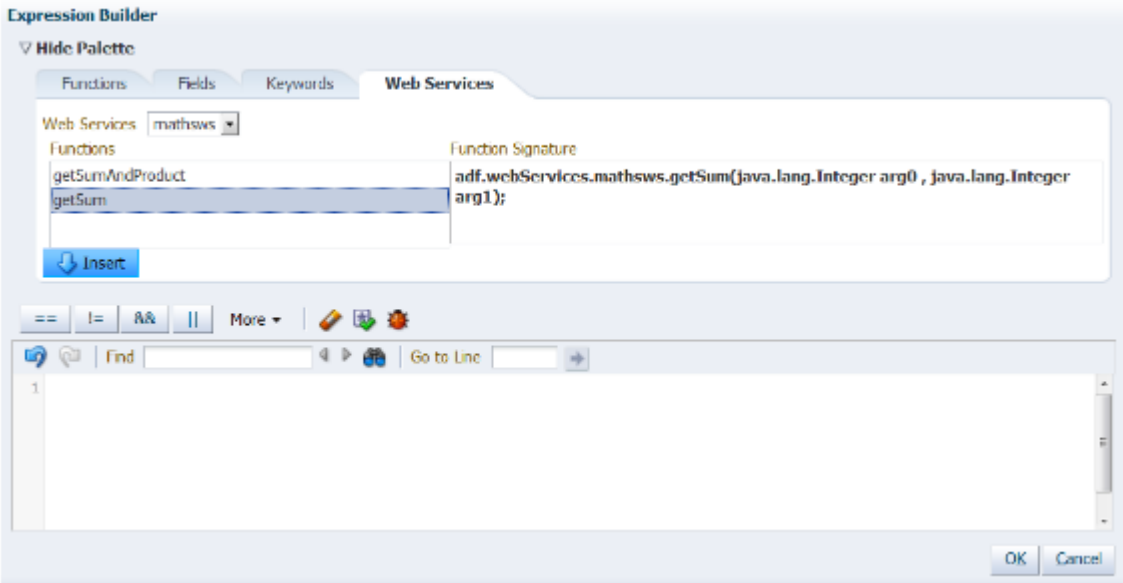
   The Web service connection is created and the name and WSDL URL are listed on the Web Services page.

Adding the Web Service Call to the Groovy Script

In the Expression Builder dialog that you see when you create or edit Groovy scripts, there is a **Web Services** tab that lists the Web services for which you have created a connection. For each Web service you can include calls to the available methods in your Groovy script.

1. In Application Composer, edit the custom field that uses the expression that will contain the Web service call.
2. Click the Expression Builder icon.
3. In the Expression Builder dialog, select the **Web Services** tab.
4. Select `mathsws` from the **Web Services** list.
5. Select **getSum** from the **Functions** list.

   The code that will be inserted is shown under **Function Signature**, as illustrated in the figure.



6. Position the cursor at the place in the script where you want to insert the Web service call.
7. Click **Insert** to insert the code to invoke the Web service method.
8. Update the script so that two integer values are provided as arguments for the Web service call.
9. Click **Submit**.
10. Verify that the Web service call succeeds; in this example the custom field should have the expected default value.

Calling an Internal Web Service from Groovy using SAML for ID Propagation : Worked Example

This example shows how to create a connection to a Fusion Applications Web service and call the Web service from a Groovy script used in Oracle Fusion CRM Application Composer. The Web service is secured by using Security Assertion Markup Language (SAML), which propagates the current user's security credentials for authentication. For this example, the Web service is used to calculate a custom field's default value.

The following table summarizes key decisions for this scenario:

Decisions to Consider	In This Example
What name will you use for the Web service connection?	mathsws
What is the URL of the Web Services Description Language (WSDL) file that you will use?	https://internal-hosted:7102/MathsWS-Model-context-root/SamlOrUsernameTokenWithMessageProtection?wsdl **Note** The URL shown here is an arbitrary example. You must obtain the real WSDL URL from the service provider,
Where will the Web service be called from?	From a Groovy script expression used to calculate a custom field's

	default value.
Which Web service method will be called from the Groovy script?	getSum This method returns the sum of two integer argument values.

To call a Web service from a Groovy script when SAML security is used, complete the following tasks:

1. Create the Web service connection.
2. Add the Web service call to the Groovy script, and verify that the call succeeds.

Prerequisites

Verify that you have completed the following prerequisite steps:

1. Get details of the WSDL URL to use from the Web service provider.
2. Create a custom field for an object that has a calculated default value.
3. Prepare the Groovy script for the expression used to calculate the field's default value. The Groovy code must prepare the argument values, which in this example are two values that are summed.

Creating the Web Service Connection

When you create a Web service connection, you specify a name for the Web service, the URL of the WSDL file, and the security scheme settings. The name is simply an identifier that is used in the list of Web services in the Expression Builder in Application Composer.

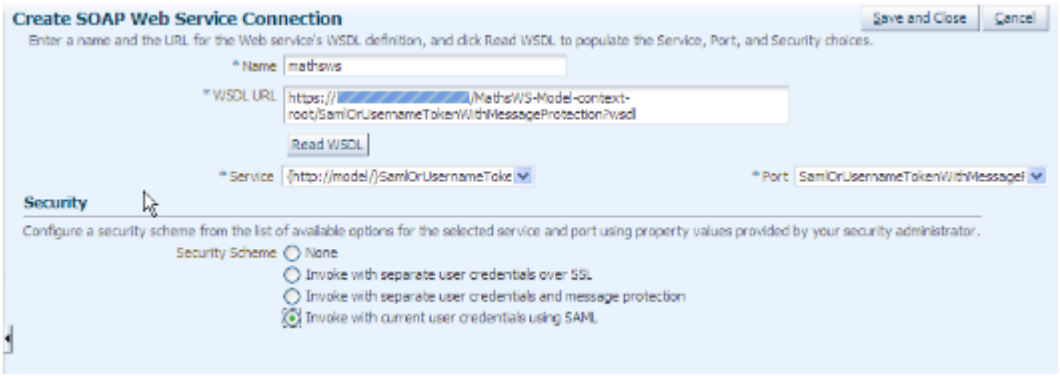
1. In Application Composer, select **Web Services** in the Common Setup pane.
2. On the Web Services page, click the New icon.
3. On the Create SOAP Web Service Connection page, enter `mathsws` in the **Name** field.

   The name must not include periods.

4. Enter `https://internal-hosted:7102/MathsWS-Model-context-root/SamlOrUsernameTokenWithMessageProtection?wsdl` in the **WSDL URL** field, and click **Read WSDL**.

   After you click **Read WSDL**, the **Service** and **Port** fields are filled according to values in the WSDL file. Under **Security Scheme**, the **Invoke with current user credentials using SAML** radio button becomes enabled and selected.

   This figure shows the Create SOAP Service Connection window.



5. Click **Save and Close**.

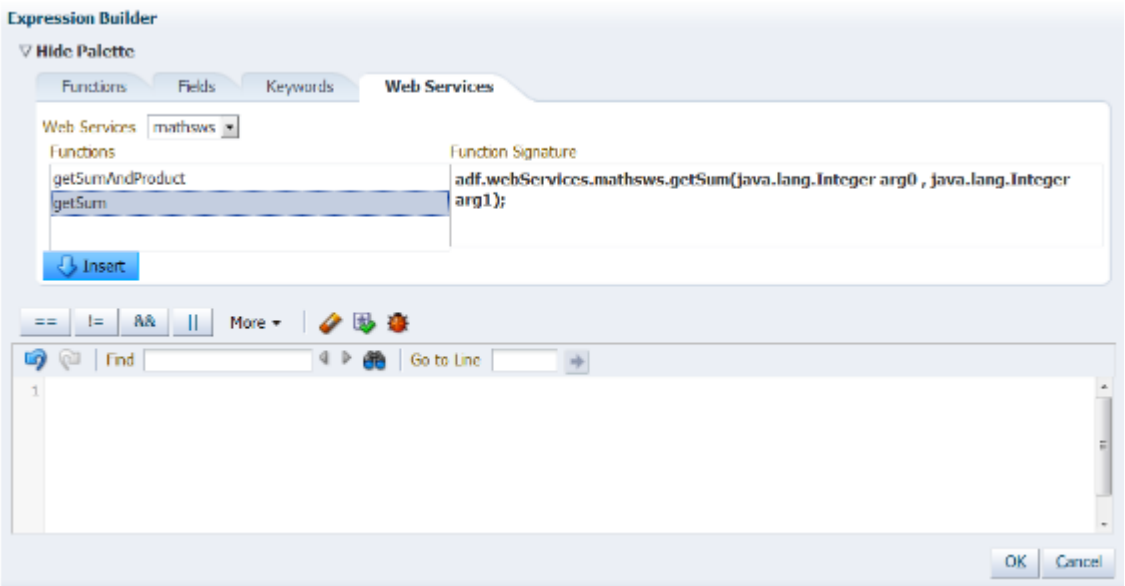
   The Web service connection is created and the name and WSDL URL are listed on the Web Services page.

Adding the Web Service Call to the Groovy Script

In the Expression Builder dialog that you see when you create or edit Groovy scripts, there is a **Web Services** tab that lists the Web services for which you have created a connection. For each Web service you can include calls to the available methods in your Groovy script.

1. In Application Composer, edit the custom field that uses the expression that will contain the Web service call.
2. Click the Expression Builder icon.
3. In the Expression Builder dialog, select the **Web Services** tab.
4. Select `mathsws` from the **Web Services** list.
5. Select **getSum** from the **Functions** list.

   The code that will be inserted is shown under **Function Signature**, as illustrated in the figure.



6. Position the cursor at the place in the script where you want to insert the Web service call.
7. Click **Insert** to insert the code to invoke the Web service method.
8. Update the script so that two integer values are provided as arguments for the Web service call.
9. Click **Submit**.

10. Verify that the Web service call succeeds; in this example the custom field should have the expected default value.

Previous Next

Home Contents Book List Contact Us