# Build AutoFill Service

Apps that provide autofill services must include a declaration that describes the implementation of the service. To specify the declaration, include a <service> element in the app manifest. The <service> element must include the following attributes and elements:

- android:name
- android:permission=BIND_AUTOFILL_SERVICE
- intent-filter with a <action> child with android.service.autofill.AutoService
- meta-data is optional

hasEnabledAutofillServices() is used by the app or service to confirm whether that app or service is enabled by the user for autofill or not.

ACTION_REQUEST_SET_AUTOFILL_SERVICE intent can be used to change AutoFillSetting.

In each request, the Android system sends an AssistStructure object to the service by calling the onFillRequest() method.
The autofill service checks whether it can satisfy the request with user data that it previously stored. If it can satisfy the request, then the service packages the data in Dataset objects.
The service calls the onSuccess() method, passing a FillResponse object that contains the Dataset objects. If the service doesn't have data to satisfy the request, it passes null to the onSuccess() method.
The service calls the onFailure() method instead if there's an error processing the request.

Example of onFillRequest

```
override fun onFillRequest(
    request: FillRequest,
```

```
    cancellationSignal: CancellationSignal,
    callback: FillCallback
) {
    // Get the structure from the request
    val context: List<FillContext> = request.fillContexts
    val structure: AssistStructure = context[context.size - 1].structure

    // Traverse the structure looking for nodes to fill out
    val parsedStructure: ParsedStructure = parseStructure(structure)

    // Fetch user data that matches the fields
    val (username: String, password: String) = fetchUserData(parsedStructure)

    // Build the presentation of the datasets
    val usernamePresentation = RemoteViews(packageName, android.R.layout.simple_list_item_1)
    usernamePresentation.setTextViewText(android.R.id.text1, "my_username")
    val passwordPresentation = RemoteViews(packageName, android.R.layout.simple_list_item_1)
    passwordPresentation.setTextViewText(android.R.id.text1, "Password for my_username")

    // Add a dataset to the response
    val fillResponse: FillResponse = FillResponse.Builder()
            .addDataset(Dataset.Builder()
                    .setValue(
                            parsedStructure.usernameId,
                            AutofillValue.forText(username),
                            usernamePresentation
                    )
                    .setValue(
                            parsedStructure.passwordId,
                            AutofillValue.forText(password),
                            passwordPresentation
                    )
                    .build())
            .build()

    // If there are no errors, call onSuccess() and pass the response
    callback.onSuccess(fillResponse)
```

```
}
data class ParsedStructure(var usernameId: AutofillId, var passwordId: AutofillId)
data class UserData(var username: String, var password: String)
```

Autofill services can navigate the ViewNode objects in the AssistStructure to retrieve the autofill data required to fulfill the request. A service can retrieve autofill data using methods of the ViewNode class, such as getAutofillId().

```
public void traverseStructure(AssistStructure structure) {
    int nodes = structure.getWindowNodeCount();

    for (int i = 0; i < nodes; i++) {
        WindowNode windowNode = structure.getWindowNodeAt(i);
        ViewNode viewNode = windowNode.getRootViewNode();
        traverseNode(viewNode);
    }
}

public void traverseNode(ViewNode viewNode) {
    if(viewNode.getAutofillHints() != null && viewNode.getAutofillHints().length > 0) {
        // If the client app provides autofill hints, you can obtain them using
        // viewNode.getAutofillHints();
    } else {
        // Or use your own heuristics to describe the contents of a view
        // using methods such as getText() or getHint()
    }

    for(int i = 0; i < viewNode.getChildCount(); i++) {
        ViewNode childNode = viewNode.getChildAt(i);
        traverseNode(childNode);
    }
```

```
}
```

To indicate that it is interested in saving the data, the service includes a SaveInfo object in the response to the fill request. The SaveInfo object contains at least the following data:

The type of user data that is saved. For a list of the available SAVE_DATA values, see SaveInfo.
The minimum set of views that need to be changed to trigger a save request. For example, a login form typically requires the user to update the username and password views to trigger a save request.

```
.setSaveInfo(
        new SaveInfo.Builder(
                SaveInfo.SAVE_DATA_TYPE_USERNAME | SaveInfo.SAVE_DATA_TYPE_PASSWORD,
                new AutofillId[] {parsedStructure.usernameId, parsedStructure.passwordId}
        )
        .build()
)

// Traverse the structure looking for data to save
traverseStructure(structure);
```

**Postpone the autofill save UI**
Starting with Android 10, if you use multiple screens to implement an autofill workflow—for example, one screen for the username field and another for the password—you can postpone the autofill save UI by using the SaveInfo.FLAG_DELAY_SAVE flag.

If this flag is set, the autofill save UI isn't triggered when the autofill context associated with the SaveInfo response is committed. Instead, you can use a separate activity within the same task to deliver future fill requests and then show the UI via a save request. For more information, see SaveInfo.FLAG_DELAY_SAVE.
**– DataSet with authentication Example**

```
RemoteViews authPresentation = new RemoteViews(getPackageName(), android.R.layout.simple_list_item_1);
authPresentation.setTextViewText(android.R.id.text1, "requires authentication");
Intent authIntent = new Intent(this, AuthActivity.class);

// Send any additional data required to complete the request
authIntent.putExtra(MY_EXTRA_DATASET_NAME, "my_dataset");
IntentSender intentSender = PendingIntent.getActivity(
        this,
        1001,
        authIntent,
        PendingIntent.FLAG_CANCEL_CURRENT
    ).getIntentSender();

// Build a FillResponse object that requires authentication
FillResponse fillResponse = new FillResponse.Builder()
    .setAuthentication(autofillIds, intentSender, authPresentation)
    .build();
```

⇒ Send Response after authentication

```
Intent intent = getIntent();

// The data sent by the service and the structure are included in the intent
String datasetName = intent.getStringExtra(MY_EXTRA_DATASET_NAME);
AssistStructure structure = intent.getParcelableExtra(EXTRA_ASSIST_STRUCTURE);
ParsedStructure parsedStructure = parseStructure(structure);
UserData userData = fetchUserData(parsedStructure);

// Build the presentation of the datasets
RemoteViews usernamePresentation = new RemoteViews(getPackageName(), android.R.layout.simple_list_item_1);
usernamePresentation.setTextViewText(android.R.id.text1, "my_username");
RemoteViews passwordPresentation = new RemoteViews(getPackageName(), android.R.layout.simple_list_item_1);
passwordPresentation.setTextViewText(android.R.id.text1, "Password for my_username");

// Add the dataset to the response
FillResponse fillResponse = new FillResponse.Builder()
    .addDataset(new Dataset.Builder()
        .setValue(parsedStructure.usernameId,
            AutofillValue.forText(userData.username), usernamePresentation)
        .setValue(parsedStructure.passwordId,
            AutofillValue.forText(userData.password), passwordPresentation)
        .build())
    .build();

Intent replyIntent = new Intent();
```

```
    // Send the data back to the service
    replyIntent.putExtra(MY_EXTRA_DATASET_NAME, datasetName);
    replyIntent.putExtra(EXTRA_AUTHENTICATION_RESULT, fillResponse);

    setResult(RESULT_OK, replyIntent);
```

---

```
    // Parse the structure and fetch payment data
    ParsedStructure parsedStructure = parseStructure(structure);
    Payment paymentData = fetchPaymentData(parsedStructure);

    // Build the presentation that shows the bank and the last four digits of the
    // credit card number, such as 'Bank-1234'
    String maskedPresentation = paymentData.bank + "-" +
       paymentData.creditCardNumber.subString(paymentData.creditCardNumber.length - 4);
    RemoteViews authPresentation = new RemoteViews(getPackageName(), android.R.layout.simple_list_item_1);
    authPresentation.setTextViewText(android.R.id.text1, maskedPresentation);

    // Prepare an intent that displays the UI that asks for the CVC
    Intent cvcIntent = new Intent(this, CvcActivity.class);
    IntentSender cvcIntentSender = PendingIntent.getActivity(
          this,
          1001,
          cvcIntent,
          PendingIntent.FLAG_CANCEL_CURRENT
    ).getIntentSender();

    // Build a FillResponse object that includes a Dataset that requires authentication
    FillResponse fillResponse = new FillResponse.Builder()
          .addDataset(new Dataset.Builder()
                // The values in the dataset are replaced by the actual
                // data once the user provides the CVC
                .setValue(parsedStructure.creditCardId, null, authPresentation)
                .setValue(parsedStructure.expDateId, null, authPresentation)
                .setAuthentication(cvcIntentSender)
                .build())
.build();
```

⇒ Send Response After authentication

```
    // Parse the structure and fetch payment data.
    ParsedStructure parsedStructure = parseStructure(structure);
    Payment paymentData = fetchPaymentData(parsedStructure);
```

```
// Build a non-null RemoteViews object to use as the presentation when
// creating the Dataset object. This presentation isn't actually used, but the
// Builder object requires a non-null presentation.
RemoteViews notUsed = new RemoteViews(getPackageName(), android.R.layout.simple_list_item_1);

// Create a dataset with the credit card number and expiration date.
Dataset responseDataset = new Dataset.Builder()
    .setValue(parsedStructure.creditCardId,
        AutofillValue.forText(paymentData.creditCardNumber), notUsed)
    .setValue(parsedStructure.expDateId,
        AutofillValue.forText(paymentData.expirationDate), notUsed)
    .build();

Intent replyIntent = new Intent();
replyIntent.putExtra(EXTRA_AUTHENTICATION_RESULT, responseDataset);
```

**Notes:**

But it is only triggered when all conditions below are met:

- The SaveInfo associated with the FillResponse is not null neither has the FLAG_DELAY_SAVE flag.
- The AutofillValues of all required views (as set by the requiredIds passed to the SaveInfo.Builder constructor are not empty.
- The AutofillValue of at least one view (be it required or optional) has changed (i.e., it's neither the same value passed in a Dataset, nor the initial value presented in the view).
- There is no Dataset in the last FillResponse that completely matches the screen state (i.e., all required and optional fields in the dataset have the same value as the fields in the screen).
- The user explicitly tapped the autofill save UI asking to save data for autofill.

**Customizing the autofill save UI**

The service can also customize some aspects of the autofill save UI:

- Add a simple subtitle by calling Builder#setDescription(CharSequence).
- Add a customized subtitle by calling Builder#setCustomDescription(CustomDescription).

- Customize the button used to reject the save request by calling Builder#setNegativeAction(int, IntentSender).
- Decide whether the UI should be shown based on the user input validation by calling Builder#setValidator(Validator).

⇒ **Complete**