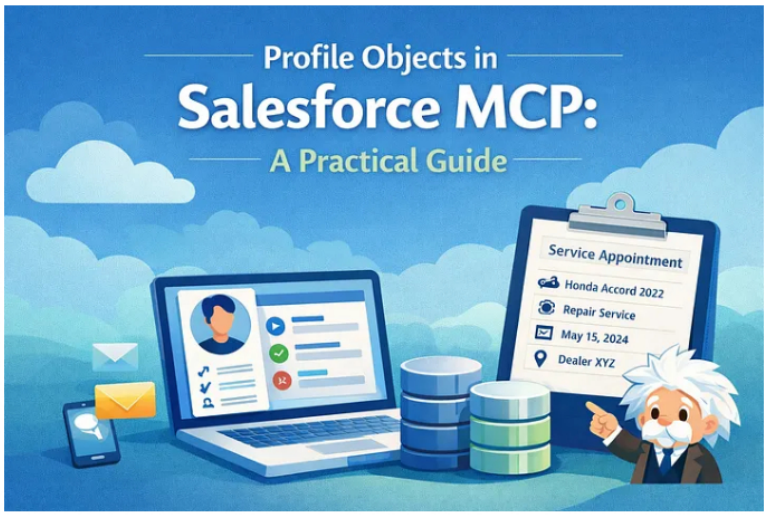
Data Capture in Profile Objects: Salesforce Marketing Cloud Personalization

CloudCove.ai Follow 5 min read · 2 days ago



Salesforce Marketing Cloud Personalization allow you to store structured, custom data about your users — beyond basic attributes like name or email. This enables powerful cross-domain personalization, targeted segmentation, and contextual messaging.

Real-World Scenario

Vehicle Service Appointment Tracking

Suppose you have a vehicle website where users can book service appointments, and your website runs on multiple domains like booking.yoursite.com, info.yoursite.com, and finance.yoursite.com. The goal is to store each user's appointment details in their profile so you can easily target users with upcoming appointments, send reminder messages before their service date, automatically fill their details when they visit again, and show their appointment history across all your websites for a more personalized experience.

Step 1: Create the Profile Object

1 Navigate to Settings

First, go to **Settings** in Salesforce Marketing Cloud Personalization, then open **Catalog and Profile Objects**, and click the **Add New (+)** button under **User Profile Object Types** to create a new profile object.

2 Define the Object Structure

Next, define the object structure by giving the object a name like **ServiceAppointment** and adding important attributes. These include

appointmentId as a unique ID for each appointment, **serviceType** to specify whether it is maintenance, repair, or check-up, **vehicleModel** to store the vehicle name such as Honda Accord 2022, **appointmentDate** to store the scheduled service date, **dealerName** to store the service center name, and **status** to track whether the appointment is scheduled, completed, or canceled.

3 Save and Enable
Finally, click **Save** to create the object and then **enable it** so it can be used for personalization. If you already have a vehicle catalog object, you can also link this profile object to it to get better context and use the data more effectively.

Pro Tip: *Think about what data you'll actually use for personalization or reporting. Don't create attributes you won't leverage — keep it action-driven.*

Step 2: Capture Data via Browser

When a user completes a service booking form and clicks "Confirm Appointment," capture the data with this client-side code:

Complete Implementation Example

```
// Service Appointment Form Submission
document.querySelector('#appointment-form').addEventListener('submit', function(
  e.preventDefault();

  // Get form data
  const formData = new FormData(this);
  const appointmentId = 'APT-' + Date.now(); // Generate unique ID

  // Send to Salesforce MCP
  SalesforceInteractions.sendEvent({
    action: "Book Service Appointment",
    user: {
      profileObjects: {
        ServiceAppointment: [
          {
            id: appointmentId,
            attributes: {
              serviceType: formData.get('serviceType'),
              vehicleModel: formData.get('vehicleModel'),
              appointmentDate: formData.get('appointmentDate'),
              dealerName: formData.get('dealerName'),
              status: "scheduled"
            }
          }
        ]
      }
    }
  });

  // Show confirmation
  alert('Appointment booked successfully! ID: ' + appointmentId);

  // Submit form to backend
  this.submit();
});
```

Updating an Existing Appointment

```
// Update appointment status (e.g., when user completes service)
function updateAppointmentStatus(appointmentId, newStatus) {

  SalesforceInteractions.sendEvent({
    action: "Update Appointment Status",
    user: {
      profileObjects: {
        ServiceAppointment: [
          {
            id: appointmentId,
            attributes: {
              status: newStatus, // "completed" or "canceled"
              completedDate: newStatus === "completed" ? new Date().toISOString(
            }
          }
        ]
      }
    }
  });

}
// Usage
updateAppointmentStatus('APT-1234567890', 'completed');
```

Best Practice: Always generate a unique id for each Profile Object entry. This allows you to update specific records later without creating duplicates.

Step 3: Use Profile Objects for Personalization

Now that appointment data is stored in user profiles, here's how to leverage it:

Segmentation

You can group users based on their service appointment data to target them more effectively. For example, you can create segments of users who have appointments in the next 7 days, users who previously booked a repair service, and users who canceled their appointments, so you can send them relevant messages or offers.

Cross-Domain Campaigns

You can recognize the same user across different websites and show personalized information. For example, when the user visits another site, you can display a message like "Welcome back! Your last appointment was with Dealer XYZ" and also show their service appointment history on other domains like finance.yoursite.com.

Triggered Messages

You can automatically send reminder messages to users based on their appointment date. For example, send a reminder 3 days before the service and another reminder one day before the appointment so users don't forget and are prepared for their scheduled service.

Contextual Offers

You can show personalized offers to users based on their service history. For example, users who recently had a repair can receive a discount on their next service, users who regularly do maintenance can be rewarded with loyalty benefits, and users who completed a service can be sent a satisfaction survey to collect their feedback.

Campaign Example: Upcoming Appointment Reminder

```
// In your sitemap or campaign logic
const upcomingAppointments = user.profileObjects.ServiceAppointment.filter(apt =
  const daysUntil = (new Date(apt.appointmentDate) - new Date()) / (1000 * 60 *
  return apt.status === "scheduled" && daysUntil >= 0 && daysUntil <= 3;
});
if (upcomingAppointments.length > 0) {
  const appointment = upcomingAppointments[0];

  // Show personalized banner
  return {
    contentZones: [
      {
        name: "hero-banner",
        content: `

Upcoming Service Appointment

Vehicle: ${appointment.vehicleModel}

Service: ${appointment.serviceType}

Date: ${formatDate(appointment.appointmentDate)}

Dealer: ${appointment.dealerName}

              View Details


        `
      }
    ]
  };
}
```

Email Campaign Example

```
// Einstein Recipe for email personalization
export class AppointmentReminderRecipe implements CampaignTemplateComponent {

  run(context: CampaignComponentContext) {
    const upcomingAppointments = context.user.profileObjects.ServiceAppointment
      .filter(apt => {
        const daysUntil = this.getDaysUntil(apt.appointmentDate);
        return apt.status === "scheduled" && daysUntil === 3;
      });

    if (upcomingAppointments.length > 0) {
      return {
        // Send reminder email 3 days before
        emailTemplate: "appointment-reminder",
        emailData: {
          appointmentDetails: upcomingAppointments[0],
          userName: context.user.attributes.firstName
        }
      };
    }

    return null;
  }

  getDaysUntil(dateString: string): number {
    return Math.ceil(
      (new Date(dateString).getTime() - new Date().getTime()) / (1000 * 60 * 60
    );
  }
}
```

What Happens Internally

Each **ServiceAppointment** is stored as a child object inside the user's profile, which means all appointment details are directly connected to that user. This data can be used in tools like Audience Builder, campaign targeting rules, Einstein Recipes, and for personalization across web, email, and mobile. The same logic works across all channels, so you don't need to create it separately for each platform. Any updates made to Profile Objects are available

for each platform. Any updates made to Profile Objects are available immediately for personalization. Profile Objects can also be connected to catalog objects for better context, and since they support multiple entries, each user can have many appointments stored, while the system automatically manages the data and makes it easy to access.

Best Practices

- Store only action-driven data you'll use for targeting or reporting
- Use clear, consistent naming conventions
- Always include a unique ID for each object entry
- Test data capture in multiple scenarios before production
- Document what each attribute represents
- Keep data updated (e.g., change status from "scheduled" to "completed")

Common Profile Object Operations

```
// 1. Add new Profile Object entry
SalesforceInteractions.sendEvent({
  user: {
    profileObjects: {
      ObjectName: [{
        id: "unique-id",
        attributes: { /* your data */ }
      }]
    }
  }
});
// 2. Update existing entry (same id)
SalesforceInteractions.sendEvent({
  user: {
    profileObjects: {
      ObjectName: [{
        id: "existing-id",
        attributes: { status: "updated" }
      }]
    }
  }
});
// 3. Access in campaigns/recipes
const userObjects = user.profileObjects.ObjectName;
const filteredObjects = userObjects.filter(obj => obj.status === "active");
// 4. Use in segmentation
// In Audience Builder: "User Profile Object → ServiceAppointment → status equal
```

Conclusion

Profile Objects make Salesforce Marketing Cloud Personalization more powerful by allowing you to store detailed and structured user data instead of just basic information. This helps you track user activities across different domains, create better audience segments for targeting, deliver more relevant and timely personalized experiences, and keep all user-related data in one place as a single source of truth.

CloudCove.ai
https://www.linkedin.com/company/cloudcove-ai

Salesforce | Marketing Cloud | Personalization | Mcp | Salesforce Interaction