If you are using AEM 6.2 Forms, you can use the pre-fill service to programmatically (using java code) pre-fill the forms. You can implement the following interface to pre-fill forms without providing the DataRef attribute:

```
package com.adobe.forms.common.service;
import java.io.InputStream;
/**
 * Interface to provide Data XML given a URL. The implementations should return the xml data given the options
 */
public interface DataXMLProvider {
    public final static String PROTOCOL_CRX="crx://";
    public final static String PROTOCOL_HTTP="http://";
    public final static String PROTOCOL_HTTPS="https://";
    public final static String PROTOCOL_FILE="file://";

    public final static String XML_PROVIDER_NAME = "dataxmlprovidername";
    public final static String XML_PROVIDER_LABEL = "dataxmlproviderlabel";
    /**
     * Returns DataXML given the dataxml options object. The implementations can decide which options to use and
     * determine the correct data XML to be returned.
     * @param options options for helping the implementations for returning the data xml
     * @return dataXML given a set of options
     * @throws FormsException
     */
    public InputStream getDataXMLForDataRef(DataXMLOptions options) throws FormsException;

    /**
     * Returns the name of the service that can be used in the SERVICE protocol. i.e. service://<service_name>/path
     * @return service name
     */
    public String getServiceName();

    /**
     * Description of the Service to be shown in the Dropdown where service selection is shown.
     * @return service description
     */
    public String getServiceDescription();
}
```

You must provide your own implementation of the getDataXMLForDataRef API. Forms runtime passes a set of options as listed below:

```
package com.adobe.forms.common.service;

import org.apache.sling.api.SlingHttpServletRequest;
import org.apache.sling.api.resource.Resource;

import java.util.Map;

/**
 * The DataXMLOptions captures the set of parameters to be passed to the DataXMLProvider for getting the data xml
 * for a form. The following options are supported currently
 * dataRef : path of the XML
 * formResource : resource of the Form for which the data is required
 * pagePath : path of the page which renders the form
 * aemFormContainer : If the Form is being embedded inside an AEM Form Container than the path of the AEM Form Container
 *                    resource. It can be null if the form is not being embedded inside a Container.
 */
public class DataXMLOptions {
    private String dataRef;
    private Resource formResource;
    private String pagePath;
    private Resource aemFormContainer;
    private String serviceName;
    private Map<String, Object> paramMap;

    /**
     * path of the XML being passed as the sling request parameter or request attribute.
     * @return path of the XML being passed as the sling request parameter or request attribute or null if nothing passed
     */
    public String getDataRef() {
        return dataRef;
    }

    /**
     * set the path of the XML from where the data has to be obtained
     * @param dataRef path of the XML from where the data has to be obtained
     * @return the same object on which this API is called. This has been done mainly for chaining multiple API calls.
     */
    public DataXMLOptions setDataRef(String dataRef) {
        this.dataRef = dataRef;
        return this;
    }

    /**
     * Form resource present inside DAM for which the data is required.
     * @return resource of the form that is being ren
```

```
     */
    public Resource getFormResource() {
        return formResource;
    }

    /**
     * Form resource present inside DAM for which the data is required.
     * @param formResource Form resource present inside DAM for which the data is required.
     * @return the same object on which this API is called. This has been done mainly for chaining multiple API calls.
     */
    public DataXMLOptions setFormResource(Resource formResource) {
        this.formResource = formResource;
        return this;
    }

    /**
     * Path of the page where the form resides. For adaptive form it will be the path of the page and for mobile form
     * it will be the path of the profile which is rendering the Form.
     * @return Path of the page where the form resides
     */
    public String getPagePath() {
        return pagePath;
    }

    /**
     * Path of the page/profile (for MF) where the form resides.
     * @param pagePath Path of the page/profile (for MF) where the form resides.
     * @return the same object on which this API is called. This has been done mainly for chaining multiple API calls.
     */
    public DataXMLOptions setPagePath(String pagePath) {
        this.pagePath = pagePath;
        return this;
    }

    /**
     * Container (if any) resource in which the form is being embedded.
     * @return container if the form is being embedded otherwise null
     */
    public Resource getAemFormContainer() {
        return aemFormContainer;
    }

    /**
     * Container (if any) resource in which the form is being embedded.
     * @param aemFormContainer Container (if any) resource in which the form is being embedded.
     * @return the same object on which this API is called. This has been done mainly for chaining multiple API calls.
     */
    public DataXMLOptions setAemFormContainer(Resource aemFormContainer) {
        this.aemFormContainer = aemFormContainer;
        return this;
    }

    /**
     * returns the name of the service to be used for getting the data xml.
     * @return name of the service or null if nothing is set
     */
    public String getServiceName() {
        return serviceName;
    }

    /**
     * set the name of the service to be used for getting the data xml. If none provided, all the available service
     * will be queried for the data
     * @param serviceName name of the service to be used for getting the data xml
     * @return the same object on which this API is called. This has been done mainly for chaining multiple API calls.
     */
    public DataXMLOptions setServiceName(String serviceName) {
        this.serviceName = serviceName;
        return this;
    }

    /**
     * Returns the parameter map
     * @return map containing the parameters
     */
    public Map getParams(){
        return paramMap;
    }

    /**
     * Sets the parameter map. This parameter map is any custom payload which has to be made
     * available to the prefill service
     * A sample example could be to fetch a map from a request attribute and set it to this params.
     * @param paramMap   map of parameters
     * @return the same object on which this API is called. This has been done mainly for chaining multiple API calls.
     */
    public DataXMLOptions setParams(Map paramMap){
        this.paramMap = paramMap;
        return this;
    }
}
```

A sample implementation of the service is as listed below. Below implementation works only with adaptive forms used inside an AEM Form component in AEM Sites.

```
package com.adobe.forms.common.service.impl;

import com.adobe.forms.common.service.DataXMLOptions;

import com.adobe.forms.common.service.DataXMLProvider;

import com.adobe.forms.common.service.FormsException;

import org.apache.felix.scr.annotations.Component;

import org.apache.felix.scr.annotations.Service;

import org.apache.sling.api.resource.Resource;

import org.apache.sling.api.resource.ResourceResolver;



import javax.jcr.RepositoryException;

import javax.jcr.Session;

import java.io.InputStream;



@Component(immediate = true, metatype = true, label = "Prefill Form using User Context")

@Service(DataXMLProvider.class)

public class UserContextDataXMLProvider implements DataXMLProvider {



    @Override

    public InputStream getDataXMLForDataRef(DataXMLOptions options) throws FormsException {

        Resource aemFormContainer = options.getAemFormContainer();

        ResourceResolver resolver = aemFormContainer.getResourceResolver();

        Session session = resolver.adaptTo(Session.class);

        String userId = session.getUserID();

        InputStream result = null;

        try {

            if (aemFormContainer != null) {

                String dataPath = "/content/usergenerated/tmp/" + userId + "/" + aemFormContainer.getPath() + "/data/";

                Resource fileResource = resolver.resolve(dataPath);

                javax.jcr.Node jcrNode = fileResource.adaptTo(javax.jcr.Node.class);
```

```
                InputStream is = jcrNode.getProperty("jcr:data").getBinary().getStream();
                result = is;
            }
        } catch (RepositoryException e) {
            throw new FormsException(e);
        }
        return result;
    }

    @Override
    public String getServiceName() {
        return "userContextXMLProvider";
    }

    @Override
    public String getServiceDescription() {
        return "User Context Prefill Service";
    }
}
```

Moreover, below is a sample implementation which returns data from a fixed node. This implementation assumes that the node has a childNode jcr:content and property jcr:data contains the xml

```
@Component(immediate = true, metatype = true, label = "Default Form Prefill service")
@Service(DataXMLProvider.class)
public class DefaultDataXMLProvider implements DataXMLProvider {
    private Logger logger = LoggerFactory.getLogger(DefaultDataXMLProvider.class);

    @Override
    public InputStream getDataXMLForDataRef(DataXMLOptions options) throws FormsException {
        Resource formResource = options.getFormResource();
        ResourceResolver resolver = formResource.getResourceResolver();
```

```
        InputStream result = null;
        try {
            String nodePath = "/path/to/my/data/"
            Resource fileResource = resolver.resolve(nodePath);
            if (fileResource instanceof NonExistingResource) {
                  return null;
            }
            javax.jcr.Node jcrNode = fileResource.adaptTo(javax.jcr.Node.class);
            javax.jcr.Node jcrContent = jcrNode.getNode("jcr:content");

            result = jcrContent.getProperty("jcr:data").getBinary().getStream();
        } catch(Exception e) {
            logger.warn("unable to read data for the dataRef " + dataRef);
            throw new FormsException(e);
        }
    }

    @Override
    public String getServiceName() {
        return "myServiceName";
    }

    @Override
    public String getServiceDescription() {
        return "My Service Description";
    }
}
```