Technical Brief: Inter-App Communication, Authentication Delegation, Services Framework

Table of Contents

1 INTER-APP COMMUNICATION 4

1.1 FEATURE OVERVIEW 4

1.2 FEATURE DESCRIPTION 4

1.2.1 Client App and Server App 4

1.2.2 Methods and Parameters 5

1.2.3 Attachments 5

1.2.4 Requests, Responses and Errors 6

1.2.5 User Experience 6

1.2.6 Connection Security 6

1.2.7 Reference Documentation 7

2 AUTHENTICATION DELEGATION 8

2.1 FEATURE OVERVIEW 8

2.2 FEATURE DESCRIPTION 8

2.2.1 Policy Control 8

2.2.2 Policy Changes 8

2.2.3 User Experience 8

*2.2.4 Container Management ..... **Error! Bookmark not defined.***

3 SERVICES FRAMEWORK 10

3.1 FEATURE OVERVIEW 10

3.2 FEATURE DESCRIPTION 10

3.2.1 Service Registration and Definition 11

3.2.2 Service Providers 12

3.2.3 Service Consumers 12

3.2.4 Public and Private Services 13

3.2.5 End Users 13

3.2.6 Limitations 14

3.2.7 Reference Documentation 14

4 APPENDIX: EXAMPLE OF SERVICES FRAMEWORK AND INTER-APP COMMUNICATION 15

4.1 SERVICE EXAMPLE 15
4.2 EXAMPLE INTERFACE DESCRIPTION 16
4.3 EXAMPLE CLIENT APP 18
4.4 EXAMPLE CLIENT IMPLEMENTATION 18
4.5 EXAMPLE ROLL-OUT 19

1 Inter-App Communication

1.1 Feature Overview

The inter-app communication feature enables secure exchange of data between two BlackBerry Dynamics apps running on the same device.

Inter-app communication is an expansion of the app data exchange capability that is available in the Secure Documents API. The Secure Documents API enables exchange of individual files. Inter-app communication enables multiple files to be sent in a single exchange, and allows for a command with parameters to be specified.

The Secure Documents API is a simple file transfer capability. Inter-app communication is a rich and secure collaboration protocol.

See also the container management diagram under Authentication Delegation, below.

1.2 Feature Description

Inter-app communication provides a data exchange mechanism that is similar to web services. The feature is based on a proprietary protocol, the BlackBerry Inter-Container Communication (ICC) protocol. ICC was created as part of the development of this feature by BlackBerry.

Under ICC, two apps on the same device can exchange data as follows (ICC terms are in italics):

1. The *client app* sends a *request* to a *method* in the *server app*.
   The request can include a number of *parameters*, and a number of *attachments*.
2. The server app receives the request, and possibly executes some processing.
3. The server app can then either:
   send a normal *response* back to the client app, or
   send an *error* response back to the client app, or
   send no response.

The terms called out in the above are discussed in more detail in the following sections.

See also the Example of Services Framework and Inter-App Communication in the appendix.

1.2.1 Client App and Server App

In an ICC interaction there is one client and one server. The client sends the ICC request to the server. The server then responds.

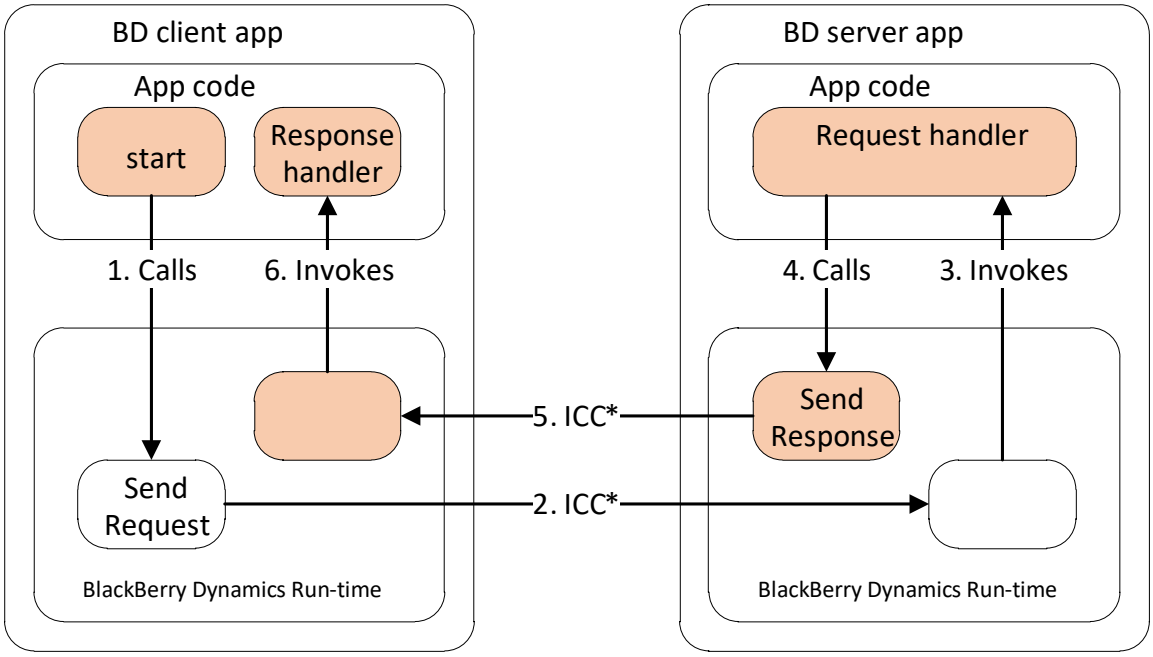
The client can discover the server by using the Services Framework, see below. Service discovery would take place prior to inter-app communication. The service discovery step is not mandatory. Without service discovery, the client has to be hard-coded with the details of the server app and is not able to use a different app that provided the same service.

The client and server apps use an inter-app communication API, as follows.

The client app contains code to send requests, and to handle responses. The server app contains code to handle received requests, and to send responses. The handlers in the client and server are invoked by the BD run-time at the appropriate points in the interaction cycle.

Communication between the apps is secured by the BlackBerry Dynamics run-time, see under Connection Security, below.

The following diagram shows the locations of the API calls, handlers, and the sequence of events. For the purposes of illustration, it has been assumed that the service sends a response, although this is optional.



*ICC: Inter-Container Communication. BlackBerry Dynamics secure communication on the device

1.2.2 Methods and Parameters

An ICC request is addressed to a named method within a service provided by a server app. A request can contain a number of named parameters, which are passed to the server as part of the ICC request. Parameters can have a numeric data-type or a textual data-type (i.e. string). An ICC response can also include a number of named parameters.

The developer of a service must provide an API of some kind that can be read by developers of clients that will use the service. The service API would list the methods of the service, describe what these methods do, and state in what circumstances the method sends a response. The API would also list the names and data types of each method's request and response parameters. If the service has specific error codes, then these would also be listed in the API.

The Services Framework, see below, includes a place for developers to publish their services' APIs.

1.2.3 Attachments

An ICC request can include attached files.

In the client side of the API, the app specifies request attachments as paths in the secure file system. The BD run-time instance in the client sends the content to the BD run-time instance in the server. The BD run-time instance in the server stores the content in the server's secure file system. The attached files are then available to the server app, in the request handler.

An ICC response can also include attached files. The content is transferred in the same way as files attached to a request and made available to the client app in the response handler.

ICC requests and responses can contain zero, one or multiple files. Files can be of any format.

1.2.4 Requests, Responses and Errors

An ICC interaction includes one request and one normal response, or one request and one error response, or one request and no response.

The client app is not required to wait for a response to one request before sending another. This means that client apps can manage their requests asynchronously. A single client could send multiple requests, to the same or different servers.

Every request has a unique ID. A response is tagged with the unique ID of the request to which it is a reply. This means that the response handler in the client can always identify to which of its open requests it is receiving a response.

If an error condition arises, then an error response will be returned instead of a normal response.

For error conditions such as the server app not being installed, or not being reachable for some other reason, an error will be returned by the ICC system. If the server could be reached, but could not satisfy the client's request, then the server will return the error, not the ICC system. There are a number of defined common error codes, for example there is a code to indicate that the request is for an unsupported method. In addition, a service can define its own error codes, for server-specific conditions, such as a request having invalid parameter values.

Common error codes are documented in the BlackBerry Dynamics API Reference. Service-specific errors will be documented as part of the service's interface definition, see under Services Framework, below.

1.2.5 User Experience

The user interface may "flip" between the client and server apps when an ICC request is in progress. If authentication delegation, see below, is not in use then the end user may have to enter their security password in the server app when the request is received.

If authentication delegation is in use, then the end user could be prompted to enter their security password in the authenticating app to start the server app. This would depend on what security policies had been set by the enterprise. For example, a policy can be set that forces the user to enter their security password whenever a BD app is brought to the foreground. The inactivity lock-out period, which is governed by security policies, may also come into play.

1.2.6 Connection Security

The ICC connection between two apps is authenticated and encrypted.

Authentication uses, at least, the mobile platform's mechanism for authentication of an app as an endpoint for communication. On iOS devices, authentication is provided as part of the openURL API.

The native mechanism guarantees that the communication endpoint is an instance of the expected app. This would guard against, for example, an app masquerading as GFE in order to receive files that were intended to be e-mailed securely.

Encryption of the connection over which ICC data exchange takes place uses Transport Layer Security (TLS). For communication between two BlackBerry Dynamics apps, the electronic certificates used for TLS will be signed by the enterprise Good Control server or the BlackBerry UEM server, depending on which management server you are using.

For further details see the BlackBerry Dynamics Security White Paper.

1.2.7 Reference Documentation

The BD library API for this feature is provided by the `GDService` and `GDServiceClient` classes.

These functions are documented in the relevant class reference in the API Reference on the BlackBerry Developers Network.

2 Authentication Delegation

2.1 Feature Overview

The authentication delegation feature enables one BlackBerry Dynamics app to hand off its authentication to another BlackBerry Dynamics app that is running on the same device.

2.2 Feature Description

Authentication delegation enables a BlackBerry Dynamics app (*delegating* app) to hand off its authentication to another app (*authenticating* app) that is running on the same device.

Every BlackBerry Dynamics app on a device has its own separate secure container. Use of authentication delegation, and inter-app communication, does not alter the container structure.

2.2.1 Policy Control

Authentication delegation is set by policy in the GC console or in the BlackBerry UEM management console, depending on which server you are using in your environment. The administrator can specify a particular authenticating app as part of a policy set. BD apps run by end users to whom the policy applies will then always delegate to the specified authenticating app.

The administrator can specify any BlackBerry Dynamics app that has been registered or published to Good Control or BlackBerry UEM as the authenticating app.

2.2.2 Policy Changes

The following policy changes are all supported by this feature:

Change of authenticating app from one app to a different app

Change from having an authenticating app to not having an authenticating app

Change from not having an authenticating app to having an authenticating app

There are circumstances in which these transitions could be problematic. For example, if the authenticating app has been uninstalled when the mobile device is off-line, and a policy change then arrives when the device is back on-line. See also the following sections.

2.2.3 User Experience

It is typically necessary for an end user to enter their security password to access a BlackBerry Dynamics app at certain times. For example, this may be necessary when the app starts, when the app resumes from background, or when the inactivity time out has expired and an idle lock is in place. All of this is governed by Good Control or BlackBerry UEM policies.

When authentication delegation is in use, the end user does not enter the security password of the delegating app. Instead, the end user authenticates in the authenticating app. The form of authentication used depends on the authenticating app.

If the **authenticating app is another BlackBerry Dynamics app** then the end user enters the security password for that app. In effect, user is able to unlock all delegating apps using the same password.

The user interface may “flip” between the delegating and authenticating apps during the authentication process. This is the same as the current enterprise SSO user experience.

A delegating app cannot be started if its authenticating app is not present on the device. If the user attempts to start a delegating app in that situation then an error message will be displayed. The error message will identify the missing app and give the user a clickable URL from which the authenticating app can be downloaded, if such a URL is available. The error message is a poor user experience, however, and best practice for administrators is for end users to be advised to install the authenticating app first.

Another point that is relevant to user experience is that if the authenticating app is uninstalled, the delegating app may become unable to communicate with the Good Control or BlackBerry UEM. This would mean that, for example, the delegating app could not retrieve updated policies, including a change to the authentication delegation policy. In general, re-installation of the original authenticating app should allow re-establishment of communication with Good Control or BlackBerry UEM.

Apart from the above differences in security password entry, the installation and initial activation of an app that delegates authentication is no different to one that does not. The end user must still enter their e-mail address and activation key.

3 Services Framework

3.1 Feature Overview

The BlackBerry Dynamics Services Framework provides for a catalogue of services, separate to the catalogue of apps. A service that is provided by one app can be used by another. This facilitates collaboration between BlackBerry Dynamics app developers.

The exchange of data involved in the actual use of a service is carried by BlackBerry Dynamics inter-app communication, see above. The services framework facilitates discovery, prior to use, and the establishment of common services that can be used and provided by a number of different developers.

The BlackBerry Dynamics services framework goes beyond what is typically offered by the native operating system.

Native frameworks allow apps to communicate and interoperate. This enables apps on a device to function as servers for other apps. For example, the iOS OpenURL mechanism enables an app to register a custom URL scheme, which allows the app to receive data from another app. OpenURL also allows the app that is sending data to check that the expected receiver is installed.

However, mobile operating system frameworks typically do not have a central location in which app developers can browse the services, and APIs, provided by other developers' apps.

3.2 Feature Description

The BlackBerry Dynamics services framework enables the creation of services, as distinct from apps. Created services are registered in the BlackBerry Dynamics Catalog.

A typical scenario for the services framework could be as follows:

1. A software developer creates a mobile app, which they sell through the relevant store or market place.
2. Later, the developer ports their app to the BlackBerry Dynamics platform in order to gain access to security conscious enterprise customers.
3. Even later, the developer decides to offer some of the capabilities of their app for use by other BlackBerry Dynamics apps. The app could then be sold as an add-on by other developers.

The developer in this scenario would create an *interface definition* for the service that their app offers. They would then register the service in the BlackBerry Dynamics Catalog and upload the interface definition. Other developers that wish to make use of the app's service would then be able to code their apps to use the service, according to the interface definition. End users would have to install both the app that provides the service, and the app that uses the service.

In the BlackBerry Dynamics API Reference and other more technical documentation, the interface definition may be referred to as the service *schema*.

There a number of possible variations on the above scenario, all of which are supported by the services framework:

- Services can be registered by any member of the BlackBerry Dynamics community, not necessarily a Partner or developer. It is not necessary for the creator of a service to have

registered an app. This enables the establishment of standard services that are not proprietary to any one provider.

- Services can have versions, in the same way that apps have versions. This accommodates changes to service definitions, such as the addition of new methods that offer new capabilities.
- An app can be registered as a provider of a service at any time, after the service is registered. The app could be new or already existing or could be a new version.
- Registered services can be used by in-house apps, as well as Partner apps.
- Services can be registered as private. A private service is only visible to the BlackBerry Dynamics organization that registered the service. See below for a definition of BlackBerry Dynamics organization.
- More than one app can be registered as providing a particular service. If the service is public, then the registered provider apps need not be from the same BlackBerry Dynamics organization. See below for a definition of BlackBerry Dynamics organization.

The terms introduced in the above description are discussed in more detail in the following sections.

See also the Example of Services Framework and Inter-App Communication in the appendix.

The term **BlackBerry Dynamics organization** in the above means an identified entity within the BlackBerry Dynamics infrastructure and community. For example, a Partner is an organization, as is an enterprise that deploys BlackBerry Dynamics in production, or an individual that deploys BlackBerry Dynamics for trial purposes.

When a new account is created for an individual on the BlackBerry Dynamics Network (BDN), a new organization is also created in the BlackBerry Dynamics infrastructure. This individual BlackBerry Dynamics organization can hold BlackBerry Dynamics assets. For example, if the BDN user creates a GC license key, then this is assigned to the corresponding organization in the infrastructure.

If the individual BDN account joins the social group of an enterprise or Partner, then the corresponding individual BlackBerry Dynamics organization is removed. Any assets that the individual may have held, such as GC server license keys or registered apps, are transferred to the organization of the social group that was joined.

Note that transfer of assets does not apply to ordinary BDN social groups such as the IT Peers group, which do not correspond to BlackBerry Dynamics organizations. Transfer of assets also does not apply when a vendor connection is created. A privileged procedure is used to create organizations for Partners and enterprises, and to flag that asset transfer applies to these groups.

Note that the organization structure already existed in BlackBerry Dynamics prior to the development of the services framework and was not changed by its introduction.

3.2.1 Service Registration and Definition

Services can be created and registered by any organization that is known in the BD community. An organization could be a Partner, an enterprise that deploys BD, BlackBerry Technology itself, or an individual that has signed up to the BlackBerry Dynamics Network.

A BD service is comprised of a number of specific methods, i.e. functions that can be called by client apps to make use of the service capabilities. Registration of a service always includes a description of the service's methods.

In the rest of this description, the organization that registers a service may be referred to as the creator. This is to avoid confusion with registration of apps as service providers, which is a separate activity.

When registering a service, the creator must provide all the following:

Service ID. The format will be similar to a BlackBerry Dynamics App ID, i.e. a reversed Internet domain, possibly followed by a number of sub-domains separated by full stops (periods), and then a service name. Service IDs must be globally unique. The Internet domain used should be one that belongs to the service creator.

Interface Description. This is an API that lists the methods that comprise the service, describes their input and output parameters, and states in what circumstances a service response should be expected. This is written by the developer in an Interface Description Language (IDL) defined by BlackBerry Technology. The IDL uses JSON syntax but is otherwise similar to the well-known Web Services Description Language (WSDL). The IDL itself can contain some, fairly sparse, documentation. Developers could publish further documentation on their own website, and link to that documentation from within the IDL.

Version ID. A service can have multiple versions, each with a different interface. Different apps, or different app versions, could offer different versions of the service.

Visibility. A service can be private or public. Private services can only be provided and consumed by apps that belong to the creator of the service.

Services are registered in the BlackBerry Dynamics Catalog. A registered service is a kind of asset within the BlackBerry Dynamics system, in the same way that GC licenses and registered apps are kinds of asset within the BlackBerry Dynamics system.

A single BlackBerry Dynamics organization can create multiple services.

3.2.2 Service Providers

Once a service has been created and registered, apps that provide the service can be identified. Identifying an app as providing a service is an additional form of registration. The developer of an app specifies which services, if any, the app provides.

The developer can specify which services an app provides when they first register the app in the BlackBerry Dynamics Catalog. The developer can also specify services when registering a new version of an app. The developer can also specify services for an already registered app or version.

If the service is private, then only apps that belong to the organization that created the service can be registered as providers. If the service is public, then any organization's apps can register as providing the service.

Note that BlackBerry Dynamics apps are not obliged to provide any services. Service-providing apps must be BlackBerry Dynamics apps that are coded to use the server side of the BlackBerry Dynamics inter-app communication API, see above.

3.2.3 Service Consumers

Once a service has been created, and at least one service-providing app has been registered, service-consuming apps can be developed.

(In theory, a service-consuming app could be developed against a service definition alone, without a service-provider being available. This is probably not practical though, since there would be no way for the developer to see the effects of calling the service.)

Service-consuming apps will be BlackBerry Dynamics apps that use the service discovery API. The service discovery API queries the BlackBerry Dynamics Catalog in the BlackBerry Dynamics NOC.

In the service discovery API, the app specifies the ID of the service that the app is coded to consume. The API returns the details of all BlackBerry Dynamics apps that provide the service, and have activated on the end user's device. The returned details include the app bundle ID, which is then used to establish a client-server inter-app communication link between the consumer and the provider.

Once the link is established, the service consumer sends method calls to the service provider using the client side of the BlackBerry Dynamics inter-app communication API, see above.

The link between consumer and provider would be terminated if one or other app was terminated, or if the device was switched off. After a link termination, the consumer app would have to repeat the service discovery process in order to re-establish the link. In theory, the link could be established to a different provider of the same service, if another provider was installed on the device.

Changes to the on-device service configuration are notified to the app layer by the BlackBerry Dynamics run-time. This would include, for example, the installation of a new or upgraded provider of the service. After receiving a notification of service configuration change, the consumer could repeat the service discovery process.

Note that it is possible to use inter-app communication without first using service discovery. In that case the app would always be a client of the same server app, specified directly by its bundle ID.

3.2.4 Public and Private Services

If a service is public then its interface definition is visible to all BlackBerry Dynamics organizations. Any organization might then choose to develop a service-consuming app.

If a service is private then its interface definition is only visible to the organization that created the service, and hence only that organization could develop a consumer.

It seems likely that only organizations with a relatively large number of apps would benefit from private service registration. For example, a Partner offering a suite of BlackBerry Dynamics apps might create a private service to facilitate communication within the suite.

3.2.5 End Users

End users must install both a service-providing app and a service-consuming app in order to use a service.

End users that install only the service consumer will not be able to use the service. When the service consumer attempts to discover the provider, discovery will fail. Similarly, if the consumer attempts to send a service request to a provider that is not installed, then the service request will fail with a specific not-found error code. In either case, the consumer app could display an error message, or show that the service is not available in some other way.

The consumer app on its own may still provide the end user with useful capabilities, depending on how the client is coded. A service-providing app could also have stand-alone capabilities that could be used outside the service.

3.2.6 Limitations

There will be no notifications from the BlackBerry Dynamics Network for:

New service creation

New provider registered for existing service

New service version registration

For example, a developer of a service consumer could not be notified by BDN when a new version of the service was registered by the creator of the service.

3.2.7 Reference Documentation

The BlackBerry Dynamics library API for this feature is provided by a function in the BlackBerry Dynamics run-time object interface class, `GDiOS`

`getAppDetailsForService:andVersion:`

This function is documented in the relevant class reference in the API Reference on BDN.

4 Appendix: Example of Services Framework and Inter-App Communication

This section gives an example of usage of the BlackBerry Dynamics Services Framework, and BD Inter-App Communication. This is for readers wanting something more concrete than the above descriptions. The example is for illustration purposes only and is not based on any known requirements or customers.

4.1 Service Example

In this example, the service is as follows:

The service adds a watermark to a supplied image or PDF. The watermark can be a styled text, or an image.

The service is called Stampit.

The Stampit service was created by the software house Fillwater. Fillwater is a registered BlackBerry Dynamics Partner.

Fillwater has developed a BlackBerry Dynamics app, also called Stampit, which is a provider of the Stampit service.

4.2 Example Interface Description

Fillwater defines their Stampit service as follows:

Service ID: com.fillwater.gd.services.stampit

Version: 1.0.0.0

Method: **watermark_text**

Request parameter	**Type**
Filename	attachment
Watermark	text
Style	number
Response parameter	**Type**
Filename	attachment

Stamps the provided file with a watermark consisting of the specified text. The text is set in a specified style, as follows:

0 – Diagonally

1 – Vertically

2 – Horizontally

The method returns the stamped file. The returned file will be in the same format as the request file.

The request file can be any of the following formats: PNG, JPEG, PDF. If the file is a PDF, the watermark will be stamped on every page.

The method returns an error if the file format is not one of those listed, or if the style specifier is out of range.

Method: **watermark_image**

Request parameter	**Type**
Filename	attachment
Watermark	attachment
Response parameter	**Type**
Filename	attachment

Stamps the provided file with a watermark consisting of the specified image. The watermark image will be treated as an overlay. Returns the stamped file. The returned file will be in the same format as the request file.

The request file can be any of the following formats: PNG, JPEG, PDF. If the file is a PDF, the watermark will be stamped on every page.

The method returns an error if either file is not a recognized format.

Fillwater has created an interface description in the BlackBerry Dynamics services IDL. The interface description has been uploaded to the BD catalogue.

Fillwater has also registered their Stampit app, with com.fillwater.gd.stampit as the BlackBerry Dynamics App ID.

4.3 Example Client App

In this example, there is a requirement for an app, as follows.

> The app is for insurance salespeople. The user can enter the age, post code and vehicle details of a prospective customer into the app. The app then generates a car insurance illustration, based on the values entered and rules synchronized from an app server.
>
> The illustration is a PDF. The PDF can be watermarked to indicate its status as Provisional or Offered.

The app will be an in-house development of the WeSure insurance company. The app is called WeSure On-The-Spot, abbreviated to OTS.

4.4 Example Client Implementation

Implementation of the example requirements could happen as follows.

WeSure select BlackBerry Dynamics as the platform to deliver OTS. This means that OTS will be a BlackBerry Dynamics app.

A software developer from WeSure registers on BDN and browses the services catalogue. The WeSure developer finds the Stampit service, which will meet their watermarking requirement.

WeSure obtain a trial license for Stampit from Fillwater and begin development of OTS. The OTS mobile app makes use of the Stampit API through the BlackBerry Dynamics services discovery and inter-app communication API.

The OTS app source includes code like the following. (Code is for illustration purposes only.)

```
// GenerateDraftFor
// OTS function to create a draft illustration, after data has been entered
- (BOOL)GenerateDraftFor: (OTSDataEntrySet*)CustomerDetails
{
    // Open a progress spinner.
    [OTSinProgress];

    // Main processing function to generate the illustration PDF file.
    NSString Illus = [GenerateIllustrationFor:CustomerDetails];

    // BD API, called to obtain a reference to the run-time object
    gdLibrary = [GDiOS sharedInstance];

    // BD service framework API
    NSArray* StampitSvc =
        [gdLibrary getAppDetailsForService:@"com.fillwater.gd.services.stampit"
                                   andVersion:@"1.0.0.0"]

    // StampitSvc should now be a single-element array, containing the GD App ID of the
    // app that provides the service, if it is installed.
    // If the service provider is not installed, draft generation fails.
    if ([StampitSvc count] < 1) {
        [OTSendProgress];
        return NO;
    }

    // Construct a service message to send to the client
    // Parameters ...
    NSDictionary* params =
        [NSDictionary dictionaryWithObjectsAndKeys:
```

```
                                                    @"DRAFT", @"Watermark",
                                                    1, @"Style",
                                                    nil];

    // Attachment ...
    NSArray* attachments = [NSArray arrayWithObjects:Illus, nil];


    // Send request
    [GDSecureContainerComms sendRequest:requestId
                              toService:@"com.fillwater.gd.services.stampit"
                             withMethod:@"watermark_text"
                             withParams:params
                         withAttachments:attachments
                           toApp:StampitSvc[0]];

    return YES;
}
```

Processing then continues in the response handler, as shown in the following. (Code is for illustration purposes only.)

```
- (void)receivedResponse:(id)requestId withCode:(int)code withValue:(id)resultValue
withMessage:(NSString*)message withAttachments:(NSArray*)attachments fromApp:(NSString*)app
{
    // Close progress indicator
    [OTSendProgress];
    if (code && message)
    {
        // alert for error
        NSLog(@"%@", message);
        UIAlertView* alertView = [[UIAlertView alloc] initWithTitle:@"Watermark Error"
                                                            message:message
                                                           delegate:nil
                                                  cancelButtonTitle:@"OK"
                                                  otherButtonTitles:nil];
        [alertView show];
    }
    else {
        // No error, show the watermarked file.
        [OTSshowPDF:attachments[0]];
    }
}
```

Note. Server app code is not shown.

4.5 Example Roll-out

To deploy their OTS app, WeSure take the following steps.

1. Obtain production BlackBerry Dynamics container licenses, from BlackBerry.
2. Obtain production Stampit license, from Fillwater.
3. Establish a vendor connection with Fillwater on BDN. (This assumes that the previous Stampit trial was run under an individual developer's account.)
4. Have Fillwater publish the Stampit app to WeSure, also on BDN.
5. Roll out the Stampit BlackBerry Dynamics app to end users, including entitlement and installation of the mobile software.

6. Roll out the WeSure On-The-Spot BlackBerry Dynamics app to end users, also including entitlement and installation of the mobile software.

Any end users that installed OTS but not Stampit would be unable to use the watermarking feature. In the OTS example code, above, the illustration PDF cannot be displayed in that scenario.