

# **BlackBerry Software Development Kit**

**Version 2.5**

**Desktop API Reference Guide**

BlackBerry Software Development Kit Version 2.5 Desktop API Reference Guide  
Last revised: 25 June 2002

Part number: PDF-04955-001

At the time of publication, this document complies with the BlackBerry Desktop Manager version 2.5.

© 2002 Research In Motion Limited. All Rights Reserved. The BlackBerry and RIM families of related marks, images and symbols are the exclusive properties of Research In Motion Limited. RIM, Research In Motion, 'Always On, Always Connected', the "envelope in motion" symbol and the BlackBerry logo are registered with the U.S. Patent and Trademark Office and may be pending or registered in other countries. All other brands, product names, company names, trademarks and service marks are the properties of their respective owners.

The handheld and/or associated software are protected by copyright, international treaties and various patents, including one or more of the following U.S. patents: 6,278,442; 6,271,605; 6,219,694; 6,075,470; 6,073,318; D445,428; D433,460; D416,256. Other patents are registered or pending in various countries around the world. Visit [www.rim.net/patents.shtml](http://www.rim.net/patents.shtml) for a current listing of applicable patents.

While every effort has been made to ensure technical accuracy, information in this document is subject to change without notice and does not represent a commitment on the part of Research In Motion Limited, or any of its subsidiaries, affiliates, agents, licensors, or resellers. There are no warranties, express or implied, with respect to the content of this document.

Research In Motion Limited  
295 Phillip Street  
Waterloo, ON N2L 3W8  
Canada

Published in Canada

# Contents

	<b>About this guide.....</b>	<b>5</b>
<b>CHAPTER 1</b>	<b>Getting started .....</b>	<b>7</b>
	Writing a desktop add-in.....	7
	Sample application .....	8
	Registration.....	8
	Enabling Desktop Manager add-ins .....	8
<b>CHAPTER 2</b>	<b>Extension API reference .....</b>	<b>11</b>
	RIM Extension API .....	11
<b>CHAPTER 3</b>	<b>Desktop API reference .....</b>	<b>15</b>
	Desktop API interfaces.....	16
	IRimDatabaseAccess .....	17
	IRimTables .....	17
	IRimTable.....	18
	IRimRecords .....	20
	IRimRecord.....	22
	IRimFields.....	24
	IRimField .....	27
	IRimProgress .....	28
	IRimLogger.....	30
	IRimUtility .....	32



# About this guide

This document explains how to use the BlackBerry Desktop application programming interface (API) to extend the BlackBerry Desktop Manager synchronization feature to access or modify custom application databases on users' handhelds.

This document contains the following information:

- overview of the API and its capabilities
- detailed reference of API members

This version of the Desktop API is compatible with the BlackBerry Desktop Manager version 2.5. You can use this API with custom C++ applications written using the BlackBerry Software Development Kit version 2.5.

## About this guide

# Chapter 1

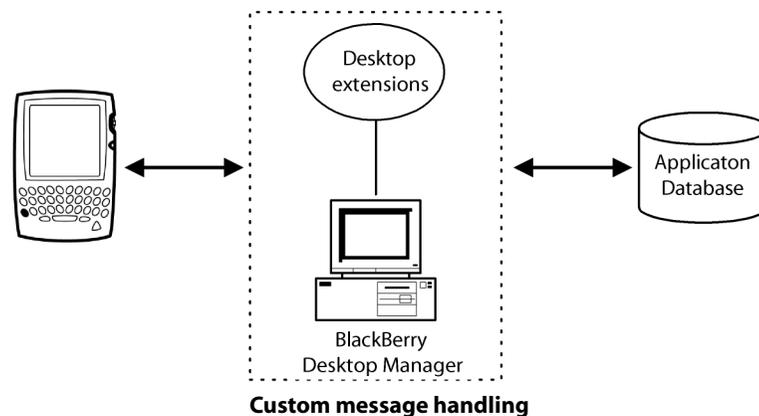
## Getting started

The BlackBerry Software Development Kit enables you to write a Component Object Model (COM) object that extends the BlackBerry Desktop Manager. This enables you to access and modify custom application databases on users' handhelds.

### Writing a desktop add-in

To add an extension to the BlackBerry Desktop Manager, you create a COM component by implementing the `IRimExtension` interface described in `RimExtension.idl`. You can write a COM object in several languages, such as C/C++ or Visual Basic. The examples provided in this guide are in C++.

Your application can import `desktopapi.tlb`. This compiled type library generates wrapper code that makes writing your application much easier.



## Sample application

For an example of how to use the Desktop API, refer to the `DesktopSample` application included with the BlackBerry SDK in `App_samples\Desktop`. This sample application writes changes to a simple contacts database on the handheld to an XML file on the desktop, `ContactList.xml`.



**Note:** The Desktop sample application included with the Desktop API requires Internet Explorer 6 for some of the XML parsing that it performs.

This sample application was created using the Microsoft Visual Studio **ALT COM AppWizard** project type. When you use this wizard to create your project, Microsoft Visual Studio automatically adds the appropriate post-build step to register your compiled DLL with the system (to see this, right-click the project, choose **Settings**, and then click the **Custom Build** tab).

## Registration

Desktop add-in applications must implement the `IRimExtension` interface. When the application is installed, it registers its object with the system. If you create your project using Microsoft Visual Studio, registration is automatically done as a post-build step after compiling.

Your application must register itself as implementing the following ID: `DFCE97AB-25ED-4335-BB00-FE5863F41DED`. This desktop add-in ID is noted in the `RimExtension.idl` file. Microsoft Visual Studio creates an `.rgs` registration file for your project, as shown in the desktop sample application. To provide the appropriate registration information, you must add the following block to the `.rgs` file:

```
'Implemented Categories'
{
    {DFCE97AB-25ED-4335-BB00-FE5863F41DED}
}
```

When the BlackBerry Desktop Manager starts, it finds desktop add-ins by retrieving the class IDs for registered objects that implement the desktop add-in category.

## Enabling Desktop Manager add-ins

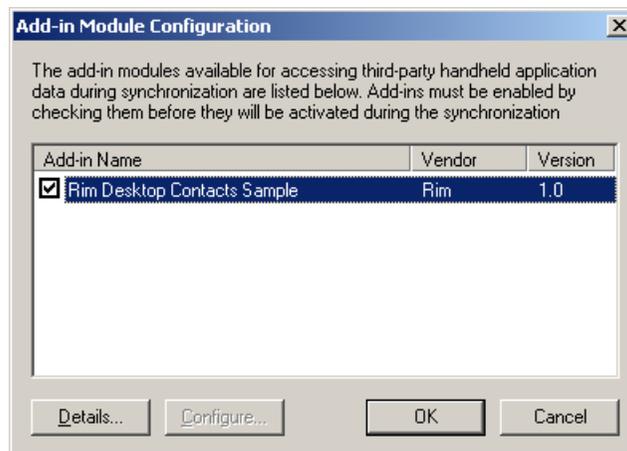
This section describes the steps that users must perform to enable add-ins in the BlackBerry Desktop Manager.

1. In the BlackBerry Desktop Manager, click the Intellisync icon to configure synchronization with their handheld. The Intellisync window appears.



**BlackBerry Desktop Manager Intellisync window**

2. In the **Synchronize now** section, select **Execute Add-in actions** to include add-ins in the synchronization process.
3. In the **Configuration** section, click **Configure Add-ins**. The Add-in Module Configuration window appears.



**BlackBerry Desktop Manager Add-in Module Configuration**

Users must select each add-in to enable it. When users then click **Configure**, your add-in can provide its additional configuration options. This is available if you include the `<hasconfiguration>true</hasconfiguration>` tag in the extension information for your add-in (refer to page 11).

## **Chapter 1: Getting started**

# Chapter 2

# Extension API

# reference

This chapter provides a detailed description of the RIM Extension API. This API provides the plug-in to the BlackBerry Desktop Manager.

## RIM Extension API

The RIM Extension API contains a single interface, `IRimExtension`, which implements `IDispatch`. Each desktop add-in must implement this interface.

`IRimExtension` contains the following functions.

### GetExtensionInfo

The `GetExtensionInfo` function retrieves an XML document that provides information for registration with the BlackBerry Desktop Manager.

```
HRESULT GetExtensionInfo( [in, out] BSTR* extensionInfo )
```

**Parameters**      `extensionInfo`      A pointer to an XML document with registration information.

**Description**      The following is an example of an extension information document:

```
<?xml version="1.0" encoding="UTF-8" ?>
<extensioninfo version="1.0">
  <vendorname>Research In Motion Limited </vendorname>
  <vendorversion>1.0</vendorversion>
```

## Chapter 2: Extension API reference

```
<path>C:\\Program Files\\Research In Motion\\BlackBerry
  SDK 2.5.0\\Desktop</path>
<description>A RIM Desktop Extension API Contacts Sample</description>
<displayname>Contacts Sample</displayname>
<hasconfiguration>true</hasconfiguration>
<access>
  <database>Contacts</database>
</access>
</extensioninfo>
```

The `<hasconfiguration>true</hasconfiguration>` tag is required if your desktop add-in provides additional configuration options for the user. This enables the **Configure** button for the add-in in the Add-in Module Configuration window (refer to page 8).

The DesktopSample application included with the BlackBerry SDK provides an example of creating the XML registration information as a string:

```
HRESULT __stdcall CDesktopSample::raw_GetExtensionInfo (
    BSTR * extensionInfo )
{
    ::SysReAllocString(extensionInfo, L"<?xml version=\"1.0\"
        encoding=\"UTF-8\" ?><extensioninfo version=\"1.0.0\">"
        L"<vendorname>Rim</vendorname>"
        L"<vendorversion>1.0</vendorversion>"
        L"<path>C:\\Program Files\\Research In Motion\\BlackBerry SDK
        2.5.0\\Desktop</path>"
        L"<description>A Rim Desktop Extension API
        Sample</description><displayname>Rim Desktop Contacts
        Sample</displayname>"
        L"<clsid>{7C23B673-FB1D-45D8-83BE-390398BA9876}</clsid>"
        L"<access><database>Contacts</database></access>"
        L"</extensioninfo>"); //the L token unicodes this string
    return 0;
}
```

## Process

The Process function is called by the BlackBerry Desktop Manager when synchronization occurs. References to an IRimUtility and IRimDatabaseAccess object are passed to the add-in.

```
HRESULT Process(
    [in] IRimUtility* pRimUtility,
    [in] IRimDatabaseAccess* pRimDeviceAccess)
```

<b>Parameters</b>	pRimUtility	A pointer to an IRimUtility object (refer to page 32).
	pRimDeviceAccess	A pointer to an IRimDatabaseAccess object, which provides access to a database on the handheld.

**See also** IRimDatabaseAccess (refer to page 17)

## Configure

This function is called by the BlackBerry Desktop Manager when the user clicks the **Configure** button for an add-in in the Add-in Module Configuration window.

```
HRESULT Configure(  
    [in] IRimUtility* pRimUtility,  
    [in] long hWnd );
```

**Parameters**

<code>pRimUtility</code>	A pointer to an <code>IRimUtility</code> object (refer to page 32)
<code>hWnd</code>	Handle to the parent window that is calling the <code>Configure</code> function.

**Description** This function is only called if the `<hasconfiguration>true</hasconfiguration>` tag was included in `extensionInfo` (refer to page 12).

**See also** `IRimUtility` (refer to page 32)

## GetErrorString

Retrieves the descriptive text associated with an error code.

```
HRESULT GetErrorString(  
    [in] int errorCode,  
    [in, out] BSTR* extensionInfo );
```

**Parameters**

<code>errorCode</code>	This parameter specifies an error code.
<code>extensionInfo</code>	This parameter is used to return the error message text.

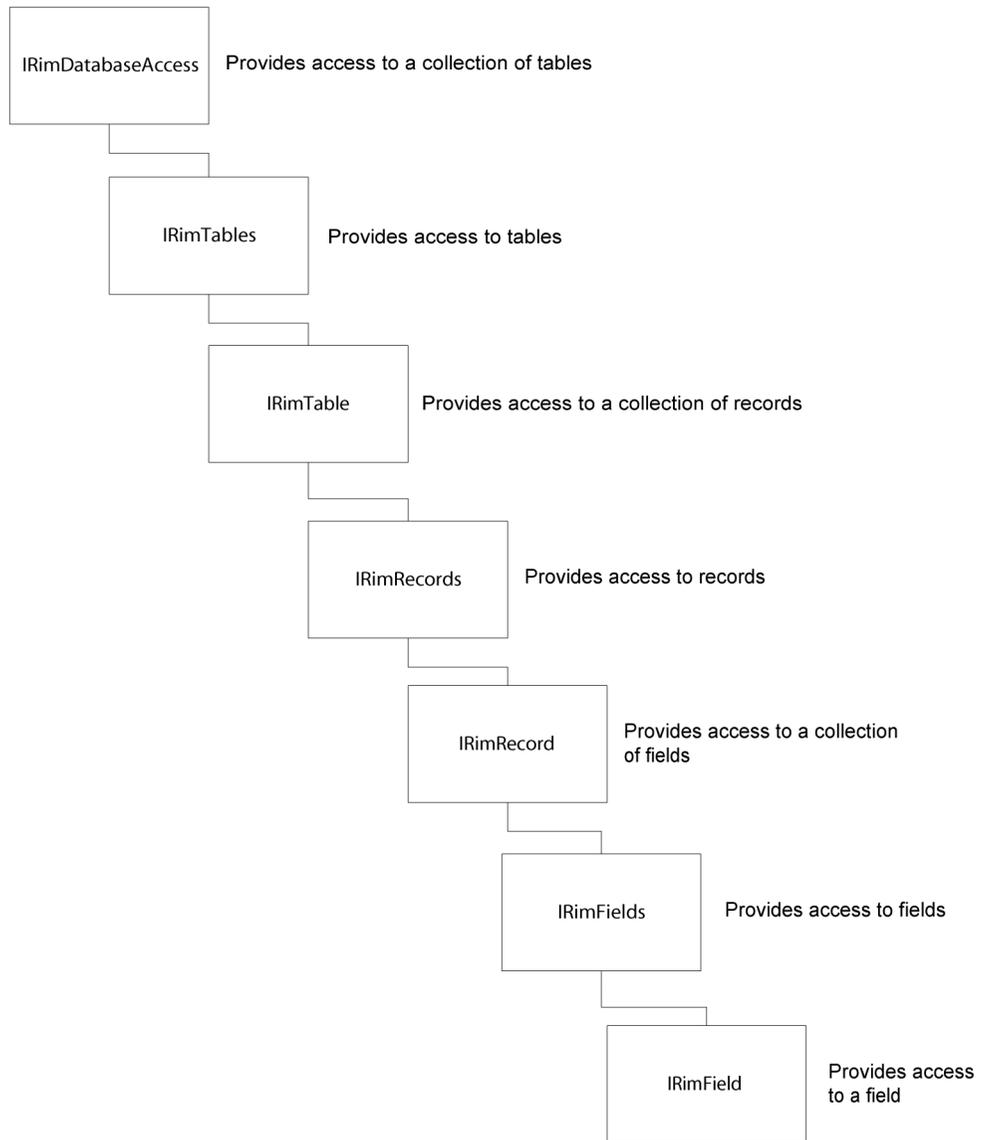
**Description** The BlackBerry Desktop Manager calls the `GetErrorString` function if the `Process()` function returns an error that is marked as interface-specific (the facility code for `HRESULT` is `FACILITY_ITF`). This means that a description can be retrieved if a desktop add-in returns an error code specific to it. The returned string can be empty.

# *Chapter 3* **Desktop API reference**

This chapter provides a detailed description of the BlackBerry Desktop API. This API is defined in the `RimDesktopApi.idl` file.

## Desktop API interfaces

The following diagram illustrates the relationship of interfaces in the Desktop API that provide access to record databases. In addition, several utility interfaces are provided to display a progress indicator, generate logs, and retrieve handheld information.



**Desktop API Interfaces**

## IRimDatabaseAccess

The IRimDatabaseAccess interface implements IDispatch. It contains the following property.

### Tables

Read-only property for a collection of tables.

```
HRESULT Tables([out, retval] IRimTables* *pVal)
```

**Parameters**    pVal                      Returns a pointer to a table collection.

## IRimTables

The IRimTables interface implements IDispatch. It represents a read-only collection of record tables requested in the <access> block of extensionInfo (refer to page 11).

### Count

Read-only property for the number of tables in the table collection.

```
HRESULT Count([out, retval] long *pVal);
```

**Parameters**    pVal                      Returns the number of tables in the collection.

### \_NewEnum

Read-only property for a new enumeration of tables.

```
HRESULT _NewEnum([out, retval] IUnknown** pVal);
```

**Parameters**    pVal                      Returns a pointer to the new enumeration.

**Description**    This function enables you to enumerate a list of items when writing scripting interfaces.

### Item

Read-only property for a table from the collection.

```
HRESULT Item([in] VARIANT var, [out, retval] IRimTable** pVal);
```

**Parameters**    var                        Specify a BSTR to retrieve the table by name or a long to retrieve the table by its index. The index value for Item is 1-based.

                  pVal                      Returns a pointer to an IRimTable object.

## IRimTable

The `IRimTable` interface implements `IDispatch`. It contains the following functions for accessing and setting record table properties.

### Id

Read-only property for the unique identifier (ID) of the table.

```
HRESULT Id([out, retval] short *pVal);
```

**Parameters**    `pVal`                      Returns the table ID.

### RecordCount

Read-only property for the number of records in a table.

```
HRESULT RecordCount([out, retval] long *pVal);
```

**Parameters**    `pVal`                      Returns the number of records in a table.

### Version

Read and write property for the table version.

```
HRESULT Version([out, retval] short *pVal);  
HRESULT Version([in] short newVal);
```

**Parameters**    `pVal`                      Returns the current table version number.  
                  `newVal`                Sets a new version number for the table.

### Name

Read-only property for the table name.

```
HRESULT Name([out, retval] BSTR *pVal);
```

**Parameters**    `pVal`                      Returns the descriptive name of the table.

## LoadRecords

Function to retrieve the records collection from the handheld.

```
HRESULT LoadRecords([in] eRIM_Mode mode,
                    [out, retval] IRimRecords * *pVal);
```

<b>Parameters</b>	mode	Specifies the appropriate action; one of the following values: <ul style="list-style-type: none"> <li>• RIM_Mode_Read — retrieves records from the handheld in read-only mode</li> <li>• RIM_Mode_Write — returns an empty records collection to which you can add a record; does not read any records from the handheld</li> <li>• RIM_Mode_ReadWrite — retrieves records from the handheld with read/write permission</li> <li>• RIM_Mode_Summary — retrieves only record summaries; must be used in conjunction with RIM_Mode_Read</li> <li>• RIM_Mode_All — retrieves record summaries from the handheld with read and write permissions</li> </ul>
	pVal	Returns a pointer to the records collection that has been retrieved from the handheld.

## Update

Function to write any changes to the handheld for records in this table; this is a bulk update.

```
HRESULT Update();
```

## Clear

Function to delete all records on the handheld for this table.

```
HRESULT Clear();
```



## Item

Property for a record in a table.

```
HRESULT Item([in] long index, [out, retval] IRimRecord* *pVal);
```

<b>Parameters</b>	index	Specifies the index of the record to retrieve. The index is 1-based.
	pVal	Returns a record.

## IRimRecord

The `IRimRecord` interface implements `IDispatch`. It contains the following functions for retrieving and setting record summary information and accessing record fields and field collections.

### Version

Read and write property for the record version.

```
HRESULT Version([out, retval] short *pVal);  
HRESULT Version([in] short newVal);
```

<b>Parameters</b>	<code>pVal</code>	Returns the version of the current record.
	<code>newVal</code>	Specifies a new version for the current record.

### RecordID

Read and write property for the record ID.

```
HRESULT RecordID([out, retval] long *pVal);  
HRESULT RecordID([in] long newVal);
```

<b>Parameters</b>	<code>pVal</code>	Returns the ID of the current record.
	<code>newVal</code>	Specifies a new ID for the current record.

### Dirty

Property for the “dirty” flag on the current record, which indicates whether or not the record has changed since it was last saved.

```
HRESULT Dirty([out, retval] BOOL *pVal);  
HRESULT Dirty([in] BOOL newVal);
```

<b>Parameters</b>	<code>pVal</code>	Returns TRUE if the record has changed since it was saved, otherwise FALSE.
	<code>newVal</code>	Specifies TRUE to indicate that the record has changed since it was last saved, or FALSE to indicate that the record has not changed.

## State

Property for the state of the record.

```
HRESULT State([out, retval] unsigned long *pVal);
HRESULT State([in] unsigned long newVal);
```

<b>Parameters</b>	pVal	Returns the state of the record.
	newVal	Specifies a new state for the record.

## Delete

Property that marks a record for deletion; the record will not be deleted until Update() is called.

```
HRESULT Delete([out, retval] BOOL *pVal);
HRESULT Delete([in] BOOL newVal);
```

<b>Parameters</b>	pVal	Returns TRUE if the record is marked for deletion; otherwise FALSE.
	newVal	Specifies TRUE to mark the record for deletion.

## Fields

Property for the fields collection in the current record.

```
HRESULT Fields([out, retval] IRimFields** pVal);
```

<b>Parameters</b>	pVal	Returns the fields collection.
-------------------	------	--------------------------------

## Load

Function to retrieve the actual record from a record summary.

```
HRESULT Load();
```

## Update

Function to write updates for a record to the handheld.

```
HRESULT Update([out, retval] long* recordID);
```

<b>Parameters</b>	recordID	Returns the ID of the record that has been updated. This is particularly useful for a new record; if no ID is specified on a new record, then an ID is generated.
-------------------	----------	---

## IRimFields

The `IRimFields` interface implements `IDispatch`. It provides the following functions for accessing and setting a collection of fields.

### Count

Property for the number of fields in the current record.

```
HRESULT Count([out, retval] long *pVal);
```

**Parameters**     `pVal`                      Returns the number of fields.

### AddField

Function to create and add a new field to a record.

```
HRESULT AddField([out, retval] IRimField** field);
```

**Parameters**     `field`                              Returns the `IRimField` object that has been added.

### Remove

Function to remove a field from the record collection. The field is not removed from the handheld until the record is updated.

```
HRESULT Remove([in] VARIANT var)
```

**Parameters**     `var`                                      Specifies the field to remove:

- `VT_I4` — `long` for the index value
- `VT_UNKNOWN` or `VT_DISPATCH` — interface reference to an `IRimField` as returned by `AddField`, `FindField`, or `Item`.

### Clear

Function to remove all fields from a collection.

```
HRESULT Clear();
```

**Description**    You must call `Update()` on the record to remove fields from the handheld.

## FindField

Function to find a record field by its field ID.

```
HRESULT FindField([in] short ID, [out,retval] IRimField** field)
```

<b>Parameters</b>	ID	Specifies the ID of the field to find.
	field	Returns the IRIMField with the specified ID.

**Description** When searching for an ID that is used in multiple IRimField objects in the collection, FindField returns the first object found in the collection with the specified ID. The same IRimField object is returned on subsequent FindField calls with the same ID. To iterate through a collection of fields with the same ID, use FindFields.

## FindFields

Function to find a collection of record fields with the same ID.

```
HRESULT FindFields([in] short id, [out, retval] IRimFields** fields)
```

<b>Parameters</b>	id	Specifies the ID of the field to find.
	fields	Returns the IRIMFields object with the specified ID.

**Description** You can use FindFields to retrieve a collection of fields with the same ID and then iterate through the collection. The following code excerpt demonstrates how to do this:

```
IRimFields *pFs;
long count;
pFields->FindFields( 1, &pFs );
pFs->get_Count(&count);
for( int i = 1; i <= count; i++ )
{
    IRimField pF = NULL;
    pFs->get_Item( i, &pF );
}
```

## \_NewEnum

Property for an enumeration of fields.

```
HRESULT _NewEnum([out, retval] IUnknown* *pVal)
```

<b>Parameters</b>	pVal	Returns a new enumeration.
-------------------	------	----------------------------

**Description** This function enables you to enumerate a list of items when writing scripting interfaces.

## Chapter 3: Desktop API reference

### Item

Property for a Field in the collection.

```
HRESULT Item([in] long index, [out, retval] IRimField **pVal)
```

<b>Parameters</b>	index	Specifies the index of the field to retrieve. The index is 1-based.
	pVal	Returns the field at the specified index.

## IRimField

The `IRimField` interface implements `IDispatch`. It contains the following functions for retrieving and setting database record fields.

### Id

Property for the field ID.

```
HRESULT Id([out, retval] short *pVal);
HRESULT Id([in] short newVal);
```

<b>Parameters</b>	<code>pVal</code>	Returns the field ID.
	<code>newVal</code>	Sets a new field ID. The valid range is 1 – 255.

### value

Property for a field value.

```
HRESULT value([out, retval] VARIANT *pVal);
HRESULT value([in] VARIANT newVal);
```

<b>Parameters</b>	<code>pVal</code>	Returns the field value as a one-dimensional, safe array of unsigned chars.
	<code>newVal</code>	Sets a new field value. Possible value types are: <ul style="list-style-type: none"> <li>• <code>VT_U11</code>   <code>VT_ARRAY</code> — one-dimensional, safe array of unsigned chars</li> <li>• <code>VT_BSTR</code> — string value (null terminator is kept)</li> <li>• <code>VT_I4</code>, <code>VT_UI4</code> — signed or unsigned long</li> <li>• <code>VT_I2</code>, <code>VT_UI2</code> — signed or unsigned short</li> <li>• <code>VT_I1</code> — signed char</li> </ul>

## IRimProgress

The IRimProgress interface implements IDiSpatch. It contains the following functions for displaying a progress indicator to the user.

### Notify

Displays a progress indicator dialog to the user.

```
HRESULT Notify(  
    [in] eRIM_Progress subfunction,  
    [in] long value1,  
    [in] long value2);
```

<b>Parameters</b>	subfunction	One of the following values: RIM_Progress_Show RIM_Progress_Hide RIM_Progress_Count RIM_Progress_Pos
	value1	Not used if subfunction is RIM_Progress_Show or RIM_Progress_Hide. If subfunction is RIM_Progress_Count, value1 is the maximum number of items that the progress indicator will handle. If subfunction is RIM_Progress_Pos, value1 is the current position.
	value2	If subfunction is RIM_Progress_Count, value2 is the step value for the progress indicator; otherwise, value2 is not used.

### getParentWindow

Retrieves a handle to the parent window for the desktop add-in application.

```
HRESULT GetParentWindow([out] long* hWnd );
```

<b>Parameters</b>	hWnd	Returns a handle to the parent window.
-------------------	------	--

## SetProgressDlgText

Sets text in a progress dialog to display to the user.

```
HRESULT SetProgressDlgText([in] eRIM_ProgressText textId, [in] BSTR text);
```

<b>Parameters</b>	textId	Specifies whether to set text for the progress dialog title or message; one of the following values: RIM_ProgressText_Title RIM_ProgressText_Msg
	text	Specifies text to display to the user in a progress dialog.

## IRimLogger

The IRimLogger interface implements IDispatch. It contains the following functions for generating log messages.

### LogStatus

Function to write a message to a log file.

```
HRESULT LogStatus([in] BSTR msg)
```

- |                    |  |                                 |
|--------------------|--|---------------------------------|
| <b>Parameters</b>  | msg  | A message to write to the logs. |
| <b>Description</b> | When LogStatus is called, the log message is logged if the LogLevel has been set to any level other than RIM_Logger_None. If the LogLevel is set to RIM_Logger_None, the function returns success but the message is not logged.<br><br>Refer to "LogLevel" on page 31 for more information. |                                 |

### LogDebug

Function to write a message to the debug file.

```
HRESULT LogDebug([in] eRIM_LogLevel level, [in] int code, [in] BSTR msg);
```

- |                    |  |   |
|--------------------|--|---|
| <b>Parameters</b>  | level  | One of the following debug levels:<br>RIM_Logger_None<br>RIM_Logger_Default<br>RIM_Logger_Verbose<br>RIM_Logger_Trace |
|                    | code   | A unique code for the debug message.  |
|                    | msg  | A debug text message.   |
| <b>Description</b> | The format for a debug message is: [timestamp]code – message.<br><br>The effect of the LogDebug function depends on which LogLevel has been set, and what value is provided for the level parameter: <ul style="list-style-type: none"><li>• When the LogLevel has been set to RIM_Logger_None, the debug message is not logged when LogDebug is called.</li><li>• When LogLevel has been set to RIM_Logger_Default, the debug message is logged unless the level parameter is RIM_Logger_None.</li><li>• When LogLevel has been set to RIM_Logger_Verbose, the debug message is logged only if the level parameter is RIM_Logger_Verbose or RIM_Logger_Trace.</li><li>• When LogLevel has been set to RIM_Logger_Trace, the debug message is logged only if the level parameter is RIM_Logger_Trace.</li></ul> Refer to "LogLevel" on page 31 for more information. |   |

## LogLevel

Property for the debug log level.

```
HRESULT LogLevel([out, retval] eRIM_LogLevel *pVal)
HRESULT LogLevel([in] eRIM_LogLevel newVal)
```

<b>Parameters</b>	pVal	Returns the current log level, one of the following: RIM_Logger_None RIM_Logger_Default RIM_Logger_Verbose RIM_Logger_Trace
	newVal	Sets a new log level as one of the following: RIM_Logger_None RIM_Logger_Default RIM_Logger_Verbose RIM_Logger_Trace

**Description** The log file is opened with an initial level of RIM\_Logger\_Default.

## IRimUtility

The `IRimUtility` interface implements `IDispatch`. It contains the following functions for retrieving handheld properties.

The class implements `IRimUtility` also implements `IRimLogger` and `IRimProgress`, so you can query the `IRimUtility` interface to access the others.

### DeviceID

Retrieves the PIN of the handheld connected to the computer serial port.

```
HRESULT DeviceID([out, retval] unsigned long *pVal);
```

**Parameters**     `pVal`                      Returns the handheld PIN.

### DeviceIDString

Retrieves the PIN, in string format, of the handheld connected to the computer serial port.

```
HRESULT DeviceIDString([out, retval] BSTR *pVal);
```

**Parameters**     `pVal`                      Returns the handheld PIN in string format.





© 2002 Research In Motion Limited  
Published in Canada