DRAFT

OpenM1 API

Open Source

Reference Guide

1.0.x

9/10

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS,WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

IWS™ OpenM1 API Reference Guide

Copyright © 2010 ImageWare Systems, Inc. All rights reserved.

OMAR-0910B

Table of Contents

1 Overview 6
2 General Approach 7
3 Standard Configuration Files 8
4 Functions 9
4.1 Standard Configuration 10
M1ReadStandard 11
M1FreeStandard 12
M1GetHeadContainer 13
M1GetContainerInfo 14
M1EnumContainerReset 16
M1EnumContainerGetNext 17
M1GetItemDefInfo 18
4.2 Data Access 20
M1CreatePackage 21
M1FreePackage 22
M1ReturnSection 23
M1SetItemString 25
M1SetItemNumber 26
M1SetItemBit 27
M1SetItemImage 28
M1GetItemString 30
M1GetItemNumber 31
M1GetItemImage 32
M1Free 33
4.3 Validation and Serialization 34
4.3.1 Validation Pass 1: Structure Adjustment 34
4.3.2 Validation Pass 2: Data Coercion 34
4.3.3 Validation Pass 3: Macro Resolution 35
4.3.4 Validation and Serialization Function 35
M1ValidatePackage 36
M1WritePackage 37
M1ReadPackage 38
5 Samples 39
5.1 OpenM1Sample1 40
5.2 OpenM1Sample2 41
5.3 OpenM1Sample3 45

5.4 OpenM1Sample4 45
5.5 OpenM1Sample5 45
Index 46

 Copyright © 2010 ImageWare Systems, Inc.

1 Overview

The OpenM1 library is designed to greatly simplify the programmatic reading, writing, and editing of biometric information packages based on the INCITS standards defined by M1.3 - Biometric Data Interchange Formats. These standards include (to name a few):

- ANSI INCITS 378-2004, Finger Minutia Format for Data Interchange
- ANSI INCITS 381-2004, Finger Image-Based Data Interchange Format
- ANSI INCITS 385-2004, Face Recognition Format for Data Interchange

OpenM1 can equally deal with the European standards counterparts, defined by the ISO JTC 1/SC 37, such as, for example:

- ISO/IEC 19794-2, Biometric Data Interchange Formats - Part 2: Finger Minutiae Data
- ISO/IEC 19794-4, Biometric Data Interchange Formats - Part 4: Finger Image Based Interchange Format
- ISO/IEC 19794-5, Biometric Data Interchange Formats - Part 5: Face Image Data

Furthermore, OpenM1 can work with any standard that defines file packages composed of multiple sections of data blocks, such as the ones above, by defining an XML configuration file for that standard following the simple OpenM1 rules.

Currently OpenM1 is available for Win32 platforms and is provided as a standard C DLL, an include file for compilation and a LIB file to link against. The library is available in 4 versions:

- Debug non-UNICODE
- Release non-UNICODE
- Debug UNICODE
- Release UNICODE

Several C++ code samples demonstrating basic functionality are also available.

 Copyright © 2010 ImageWare Systems, Inc.

2 General Approach

The OpenM1 approach is founded on the realization that most file-format standards are based on the same concept: they are formed of multiple sections containing items of information and possibly other sections.

In general, we define an **M1Item** as a container of data, and an **M1Section** as a container of other M1Sections and M1Items. We define an **M1Container** as being either an M1Section or an M1Item.

In programming jargon, M1Section and M1Item both inherit from M1Container. Hence a data package can have the following structure, for example:

```
M1Section
    M1Section
        M1Item
        M1Item
        M1Section
            M1Item
    M1Section
        M1Item
    M1Item
```

In this example, the main section is made up of 3 containers, of which the first two are sections and the last is an item. The first of these two sections contains three containers, namely two items and a section, which in turn contains only an item. The second of these two sections also only contains an item.

By using the abstraction of the container, we can easily navigate any tree structure by enumerating all containers one level below the current container. If we determine the container is a section we recurse into it, and if we determine the container is an item we can deal with the data it contains.

This recursion over containers is the foundation of the OpenM1 API.

 Copyright © 2010 ImageWare Systems, Inc.

3 Standard Configuration Files

The format of the standard configuration files are described in XML, based on the schema file **BDIF.xsd**. It mimics the nested list of sections and items in XML by using <section> and <item> elements. Given the scenario in Chapter 2, we would have the following general XML structure:

```
<section>
    <section>
        <item>
        </item>
        <item>
        </item>
        <section>
            <item>
            </item>
        </section>
    <section>
        <item>
        </item>
    </section>
    <item>
    </item>
</section>
```

On top of this structure, these XML elements also have child elements, which describe the sections and items. These elements include rules such as:

- Sections and items must have a <name>
- Sections and items can have a <description>
- Sections and items can have a <minoccurrences> as well as a <maxoccurrences>, as a means to describe that they can appear multiple times in sequence in a standard package. Lacking these elements it is assumed these containers occur exactly once.
- Sections and items can have a <reserved> element. If set to true, the OpenM1 caller creating a package need not provide data for this item or section: OpenM1 sets reserved containers automatically.
- Items must have a <format> describing the item's datatype, which can be bit, byte, word, dword, tword, string, binary, jpegorjpeg2000 or image.
- Items can have a <min> and <max> value, to be applied to numerical data, or in the case of a string or a binary datatype, these values will describe the length of the data.
- Items can have default values, specified by using the <default> element.
- Items can have a set of values that the value must take on its value from. This is specified by using a series of <listitem>s containing the allowed <name>/<value> pairs.

To define this tree structure in memory we use a nested list of **M1ContainerDef** objects. An M1ContainerDef is an abstract type for an **M1SectionDef** or an **M1ItemDef**. These objects mirror the M1Container, M1Section and M1Item types described in *Chapter 2: General Approach* on page 7. In essence ,the types defined above are intended to contain the data, whereas the *Def types are meant to define the data format.

The XML standard configuration file can also contain macros such as length() and occurrences() which enable the standard to define fields which contain the number of bytes or the number of occurrences of particular containers. This facilitates the creation of packages for the user of OpenM1 since these fields are automatically set by OpenM1.

 Copyright © 2010 ImageWare Systems, Inc.

4 Functions

The OpenM1 API provides access to the following functions.

Standard Configuration

- M1ReadStandard
- M1FreeStandard
- M1GetHeadContainer
- M1GetContainerInfo
- M1EnumContainerReset
- M1EnumContainerGetNext
- M1GetItemDefInfo

Data Access

- M1CreatePackage
- M1FreePackage
- M1ReturnSection
- M1SetItemString
- M1SetItemNumber
- M1SetItemBit
- M1SetItemImage
- M1GetItemString
- M1GetItemNumber
- M1GetItemImage
- M1Free

Validation and Serialization

- M1ValidatePackage
- M1WritePackage
- M1ReadPackage

 Copyright © 2010 ImageWare Systems, Inc.

4.1 Standard Configuration

The following functions enable reading and verifying an XML standard configuration file and enumerating the complete structure and all properties within.

Function	Description
M1ReadStandard	Reads a standard configuration file into memory.
M1FreeStandard	Frees the M1Standard object allocated by a previous call to the M1ReadStandard function.
M1GetHeadContainer	Returns the top-level container definition for a specified M1Standard object.
M1GetContainerInfo	Gets information about the specified container definition.
M1EnumContainerReset	Resets the internal enumeration counter of a container definition. This function is used in conjunction with the M1EnumContainerGetNext function.
M1EnumContainerGetNext	Iterates through all container definitions within a specified container definition.
M1GetItemDefInfo	Gets information about an M1 item definition.

For an example of the use of these functions, see *OpenM1Sample1* on page 40.

 Copyright © 2010 ImageWare Systems, Inc.

M1ReadStandard

Reads a standard configuration file into memory.

Prototype

```
int M1ReadStandard(
     const TCHAR* szPath,
     M1Standard** ppStandard,
     int nMaxError,
     TCHAR* szError
);
```

Parameters

szPath	Input parameter. Specifies the input path of the standard configuration to be loaded.
ppStandard	Output parameter. Returns a pointer to the newly-created M1Standard object.
nMaxError	Input parameter. Specifies the maximum number of characters for an error message.
szError	Input parameter. Specifies the preallocated space to receive any error message.

Discussion

The XML standard configuration file is parsed and read into memory. If any XML parsing errors occur, a detailed error message is returned. The M1FreeStandard function must be used to free the standard.

Example

Return Codes

M1_OK	Indicates that the configuration file was successfully read in to memory.
M1_FAIL	Indicates that the operation failed. Refer to the error returned for more detail.
M1_INVALIDPARAM	Indicates that one of the pointer parameters was null.
M1_MEMORYFAULT	Indicates that a memory fault occurred.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

M1FreeStandard

 Copyright © 2010 ImageWare Systems, Inc.

M1FreeStandard

Frees the M1Standard object allocated by a previous call to the M1ReadStandard function.

Prototype

```
int M1FreeStandard(
     M1Standard* pStandard
);
```

Parameters

pStandard	Input parameter. Specifies a pointer to the M1Standard object to be released.

Discussion

This function should be called after the M1ReadStandard function to free the memory buffer from that function.

Return Codes

M1_OK	Indicates that the operation was completed successfully.
M1_INVALIDPARAM	Indicates that the pStandard parameter was not specified.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

M1ReadStandard

 Copyright © 2010 ImageWare Systems, Inc.

M1GetHeadContainer

Returns the top-level container definition for a specified M1Standard object.

Prototype

```
int M1GetHeadContainer(
     M1Standard* pStandard,
     M1ContainerDef** ppContainer
);
```

Parameters

pStandard	Input parameter. Specifies a pointer to an M1Standard object.
ppContainer	Output parameter. Specifies a pointer to the container definition.

Discussion

This is the first step to enumerating all information within a given standard. Note that the container definition does not need to be released separately; it will get released with the standard via the call to M1FreeStandard function.

Return Codes

M1_OK	Indicates that the operation was completed successfully.
M1_INVALIDPARAM	Indicates that one of the pointer parameters was null.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

M1FreeStandard

 Copyright © 2010 ImageWare Systems, Inc.

M1GetContainerInfo

Gets information about the specified container definition.

Prototype

```
int M1GetContainerInfo(
     M1ContainerDef*    pContainer,
     bool* pbIsItemDef,
     TCHAR** pszName,
     TCHAR** pszDescription,
     TCHAR** pszMinOccurrences,
     TCHAR** pszMaxOccurrences,
     bool *pbIsReserved
);
```

Parameters

pContainer	Input parameter. Specifies a pointer to an M1 container definition.
pbIsItemDef	Output parameter. REturns a Boolean value that indicates the type of container definition. • true Indicates that the container definition is an M1 item definition • false Indicates that the specified definition is an M1 section definition
pszName	Output parameter. Returns the name of the container.
pszDescription	Output parameter. Returns the description of the container.
pszMinOccurrences	Output parameter. Returns the minimum number of times the container can occur in a standard configuration file.
pszMaxOccurrences	Output parameter. Returns the maximum number of times the container can occur in a standard configuration file.
pbIsReserved	Output parameter. Returns a Boolean value that indicates whether the container definition is reserved. • true Indicates that the container definition is reserved • false Indicates that the container definition is not reserved

Discussion

This function returns all information that is common to both M1 section definitions and M1 item definitions. Note that all strings are preallocated by OpenM1 and will get released with the

 Copyright © 2010 ImageWare Systems, Inc.

standard when the M1FreeStandard function is called..

Return Codes

M1_OK	Indicates that the operation was completed successfully.
M1_INVALIDPARAM	Indicates that one of the pointer parameters was null.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

M1FreeStandard

 Copyright © 2010 ImageWare Systems, Inc.

M1EnumContainerReset

Resets the internal enumeration counter of a container definition. This function is used in conjunction with the M1EnumContainerGetNext function.

Prototype

```
int M1EnumContainerReset(
     M1ContainerDef* pContainer
);
```

Parameters

pContainer	Input parameter. Specifies a pointer to an M1 container definition.

Discussion

After determining that a container definition represents an M1 section definition with the M1GetContainerInfo function, call this function to initiate iteration across all container definitions within the section container that is specified in the pContainer parameter.

Return Codes

M1_OK	Indicates that the operation was completed successfully.
M1_INVALIDPARAM	Indicates that the pContainer parameter was null.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

M1EnumContainerGetNext

M1GetContainerInfo

 Copyright © 2010 ImageWare Systems, Inc.

M1EnumContainerGetNext

Iterates through all container definitions within a specified container definition.

Prototype

```
int M1EnumContainerGetNext(
   M1ContainerDef* pContainer,
   M1ContainerDef** ppContainerNext
);
```

Parameters

pContainer	Input parameter. Specifies a pointer to an M1 container definition representing a M1 section definition.
ppContainerNext	Output parameter. Returns the next container definition in the iteration.

Discussion

Before calling this function, you must call the M1EnumContainerReset function for the M1 section definition specified in the pContainer function to reset its internal iterator. Then, typically this function is called iteratively until it returns an M1_FAIL code, which indicates the end of the list.

Return Codes

M1_OK	Indicates that the operation was completed successfully. The returned ppContainerNext parameter points to the next container definition.
M1_FAIL	Indicates that the iteration has reached the end of the list of container definitions within the specified container definition.
M1_INVALIDPARAM	Indicates that the pContainer parameter was null or represents an M1 item definition instead of a section definition.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

M1EnumContainerReset

 Copyright © 2010 ImageWare Systems, Inc.

M1GetItemDefInfo

Gets information about an M1 item definition.

Prototype

```
int M1GetItemDefInfo(
     M1ContainerDef* pContainer,
     M1ItemFormat* pItemFormat,
     TCHAR** pszMinValue,
     TCHAR** pszMaxValue,
     TCHAR** pszDefaultValue
);
```

Parameters

pContainer	Input parameter. Specifies a pointer to an M1 container definition representing a M1 item definition.
pItemFormat	Output parameter. Returns the format of the item: bit, byte, word, dword, tword, string, binary, jpegorjpeg2000 or image.
pszMinValue	Output parameter. Returns the minimum value for numeric data, or in the case of a string or a binary datatype, the minimum length of the data.
pszMaxValue	Output parameter. Returns the maximum value for numeric data, or in the case of a string or a binary datatype, the maximum length of the data..
pszDefaultValue	Output parameter. Returns the cefault value of the item.

Discussion

After determining that a container definition represents an M1 item definition with the M1GetContainerInfo function, call this function to retrieve the item definition, such as its format and optional minimum, maximum, or default values.

Return Codes

M1_OK	Indicates that the operation was completed successfully.
M1_INVALIDPARAM	Indicates that the pContainer parameter was null or represents an M1 section definition.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

 Copyright © 2010 ImageWare Systems, Inc.

M1GetContainerInfo

 Copyright © 2010 ImageWare Systems, Inc.

4.2 Data Access

The OpenM1 contains several data access functions that allow a data package to be created from scratch, and then validated and written to a file. It also allows a file to be loaded from an existing data package, with subsequent enumeration of all its data. Packages can also be modified and re-validate and re-saved, allowing for full editing functionality.

The OpenM1 API contains the following data access functions:

Function	Description
M1CreatePackage	Allocates space for a new data package and returns a pointer to its root section.
M1FreePackage	Frees a package that was allocated with the M1CreatePackage function.
M1ReturnSection	Returns a pointer to an existing or newly-created child section within the specified parent section of a configuration file.
M1SetItemString	Sets a named item within the provided section with a string value.
M1SetItemNumber	Sets a named item within the provided section with a dword value.
M1SetItemBit	Sets a named item within the provided section with a dword value of 0 or 1, based on the provided Boolean parameter.
M1SetItemImage	Sets a named item within the provided section with a binary value.
M1GetItemString	Returns the string stored in the section's item specified by name and index.
M1GetItemNumber	Returns the number stored in the section's item specified by name and index.
M1GetItemImage	Returns the image stored in the section's item specified by name and index.
M1Free	Frees a generic OpenM1 memory block. Currently only to be used on memory allocated by the M1GetItemImage function.

 Copyright © 2010 ImageWare Systems, Inc.

M1CreatePackage

Allocates space for a new data package and returns a pointer to its root section.

Prototype

```
int M1CreatePackage(
     const TCHAR* szPackageName,
     M1Section** ppHeadSection
);
```

Parameters

szPackageName	Input parameter. Specifies the name of the package to be created.
ppHeadSection	Output parameter. Returns the root section of the standard configuration file.

Discussion

The specified package name should match the name of the the standard you want to create the package for (for example, INCITS 385-2004). To avoid memory leaks, the returned head section must eventually be freed with a call to the M1FreePackage function.

Return Codes

M1_OK	Indicates that the operation was completed successfully.
M1_INVALIDPARAM	Indicates that a pointer parameter was null.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

M1FreePackage

 Copyright © 2010 ImageWare Systems, Inc.

M1FreePackage

Frees a package that was allocated with the M1CreatePackage function.

Prototype

```
int M1FreePackage(
     M1Section* pHeadSection
);
```

Parameters

pHeadSection	Input parameter. Specifies a pointer to the package that as returned by the M1CreatePackage function.

Discussion

This function should be called after the M1CreatePackage function to free the memory buffer from that function.

Return Codes

M1_OK	Indicates that the operation was completed successfully.
M1_INVALIDPARAM	Indicates that a pointer parameter was null.
M1_MEMORYFAULT	Indicates that an invalid or previously freed pointer was passed to the function.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

M1CreatePackage

 Copyright © 2010 ImageWare Systems, Inc.

M1ReturnSection

Returns a pointer to an existing or newly-created child section within the specified parent section of a configuration file.

Prototype

```
int M1ReturnSection(
     M1Section* pParentSection,
     const TCHAR* szSectionName,
     int iSection,
     M1Section** ppSection
);
```

Parameters

pParentSection	Input parameter. Specifies a pointer to an existing parent section of a configuration file.
szSectionName	Input parameter. Specifies the name of the child section to be returned.
iSection	Input parameter. Specifies the index of the child section to be returned.
ppSection	Output parameter. Returns a pointer to the child section specified.

Discussion

This function always returns a child section with the given section name and section index, regardless of whether the section already exists.

If the section already exists as a child of the parent section, then a pointer to that section is returned and the function returns M1_OK.

If the section does not exist, then the section is created and a pointer to that section is returned, and the function returns M1_SECTIONNOTEXIST. This means that both M1_OK and M1_SECTIONNOTEXIST are successful return values.

The section should never be freed. It will be deallocated when the root section is freed with the M1FreeSection function.

Note that all indexes in the OpenM1 API are 1-based. When specifying multiple occurrences of sections, the indexes should increase sequentially.

Return Codes

M1_OK	Indicates that the section already existed and that a pointer to that section was successfully returned.
M1_SECTIONNOTEXIST	Indicates that the section did not exist previously, was then created, and that a pointer to that section was successfully returned.

 Copyright © 2010 ImageWare Systems, Inc.

M1_INVALIDPARAM	Indicates that a pointer parameter was null.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

None.

 Copyright © 2010 ImageWare Systems, Inc.

M1SetItemString

Sets a named item within the provided section with a string value.

Prototype

```
int M1SetItemString(
     M1Section* pSection,
     const TCHAR* szItemName,
     int iItem,
     const TCHAR* szValue
);
```

Parameters

pSection	Input parameter. Specifies a pointer to an existing section of a configuration file.
szItemName	Input parameter. Specifies the name of the child section.
iItem	Input parameter. Specifies the index of the child section.
szValue	Input parameter. Specifies the string value to which the item should be set.

Discussion

Assuming all parameters are valid, this function always sets the value, regardless of whether the named item already exists.

If the item already exists as a child of the section, then its existing value is changed to the string provided.

If the item does not exist, the item is created and its value set as specified.

Note that all indexes in the OpenM1 API are 1-based. When specifying multiple occurrences of sections, the indexes should increase sequentially.

Return Codes

M1_OK	Indicates that the operation was completed successfully.
M1_INVALIDPARAM	Indicates that a pointer parameter was null.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

None.

 Copyright © 2010 ImageWare Systems, Inc.

M1SetItemNumber

Sets a named item within the provided section with a dword value.

Prototype

```
int M1SetItemNumber(
     M1Section* pSection,
     const TCHAR* szItemName,
     int iItem,
     int iValue
);
```

Parameters

pSection	Input parameter. Specifies a pointer to an existing section of a configuration file.
szItemName	Input parameter. Specifies the name of the child section.
iItem	Input parameter. Specifies the index of the child section.
iValue	Input parameter. Specifies the dword value to which the item should be set.

Discussion

Assuming all parameters are valid, this function always sets the value, regardless of whether the named item already exists.

If the item already exists as a child of the section, then its existing value is changed to the string provided.

If the item does not exist, the item is created and its value set as specified.

Note that all indexes in the OpenM1 API are 1-based. When specifying multiple occurrences of sections, the indexes should increase sequentially.

Return Codes

M1_OK	Indicates that the operation was completed successfully.
M1_INVALIDPARAM	Indicates that a pointer parameter was null.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

None.

 Copyright © 2010 ImageWare Systems, Inc.

M1SetItemBit

Sets a named item within the provided section with a dword value of 0 or 1, based on the provided Boolean parameter.

Prototype

```
int M1SetItemBit(
     M1Section* pSection,
     const TCHAR* szItemName,
     int iItem,
     bool bValue
);
```

Parameters

pSection	Input parameter. Specifies a pointer to an existing section of a configuration file.
szItemName	Input parameter. Specifies the name of the child section.
iItem	Input parameter. Specifies the index of the child section.
bValue	Input parameter. Specifies the Boolean value to which the item should be set.

Discussion

This function calls M1SetItemNumber and sets the iValue parameter to either 1 or 0.

Return Codes

M1_OK	Indicates that the operation was completed successfully.
M1_INVALIDPARAM	Indicates that a pointer parameter was null.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

M1SetItemNumber

 Copyright © 2010 ImageWare Systems, Inc.

M1SetItemImage

Sets a named item within the provided section with a binary value.

Prototype

```
int M1SetItemImage(
     M1Section* pSection,
     const TCHAR*  szItemName,
     int iItem,
     BYTE* pImage,
     int iImageLen
);
```

Parameters

pSection	Input parameter. Specifies a pointer to an existing section of a configuration file.
szItemName	Input parameter. Specifies the name of the child section.
iItem	Input parameter. Specifies the index of the child section.
pImage	Input parameter. Specifies the input image to set to the item.
iImageLength	Input parameter. Specifies an integer representing the input length of the image.

Discussion

Assuming all parameters are valid, this function always sets the value, regardless of whether the named item already exists. The fucntion makes a copy of the image, which gets freed when the root section gets freed via the M1FreePackage function.

If the item already exists as a child of the section, then its existing value is changed to the image provided.

If the item does not exist, the item is created and the image set as specified.

Note that all indexes in the OpenM1 API are 1-based. When specifying multiple occurrences of sections, the indexes should increase sequentially.

Return Codes

M1_OK	Indicates that the operation was completed successfully.
M1_INVALIDPARAM	Indicates that a pointer parameter was null.
M1_EXCEPTION	Indicates that an unexpected error occurred.

 Copyright © 2010 ImageWare Systems, Inc.

See Also

None.

 Copyright © 2010 ImageWare Systems, Inc.

M1GetItemString

Returns the string stored in the section's item specified by name and index.

Prototype

```
int M1GetItemString(
     M1Section* pSection,
     const TCHAR* szItemName,
     int iItem,
     TCHAR** pszValue
);
```

Parameters

pSection	Input parameter. Specifies a pointer to an existing section of a configuration file.
szItemName	Input parameter. Specifies the name of the child section.
iItem	Input parameter. Specifies the index of the child section.
pszValue	Output parameter. Returns a pointer to the string for the item.

Discussion

If the item exists, this function returns the pre-allocated string in the pszValue parameter and returns M1_OK.

If the item does not exist, or if it exists but is not stored as a string, this function returns M1_ITEMNOTEXIST. It is the responsibility of the caller to know the general type of the item's data.

Return Codes

M1_OK	Indicates that the operation was completed successfully.
M1_ITEMNOTEXIST	Indicates that the specified section does not contain an item of this datatype with that name or index.
M1_INVALIDPARAM	Indicates that a pointer parameter was null.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

None.

 Copyright © 2010 ImageWare Systems, Inc.

M1GetItemNumber

Returns the number stored in the section's item specified by name and index.

Prototype

```
int M1GetItemNumber(
      M1Section*        pSection,
      const TCHAR*            szItemName,
      int                     iItem,
      int*                    piValue
);
```

Parameters

pSection	Input parameter. Specifies a pointer to an existing section of a configuration file.
szItemName	Input parameter. Specifies the name of the child section.
iItem	Input parameter. Specifies the index of the child section.
piValue	Output parameter. Returns a pointer to the number for the item.

Discussion

If the item exists, this function returns the number in the piValue parameter and returns M1_OK.

If the item does not exist, or if it exists but is not stored as a number, this function returns M1_ITEMNOTEXIST. It is the responsibility of the caller to know the general type of the item's data.

Return Codes

M1_OK	Indicates that the operation was completed successfully.
M1_INVALIDPARAM	Indicates that a pointer parameter was null.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

None.

 Copyright © 2010 ImageWare Systems, Inc.

M1GetItemImage

Returns the image stored in the section's item specified by name and index.

Prototype

```
int M1GetItemImage(
     M1Section* pSection,
     const TCHAR* szItemName,
     int iItem,
     BYTE** ppImage,
     int* piImageLen
);
```

Parameters

pSection	Input parameter. Specifies a pointer to an existing section of a configuration file.
szItemName	Input parameter. Specifies the name of the child section.
iItem	Input parameter. Specifies the index of the child section.
ppImage	Output parameter. Returns a pointer to the image for thie item.
pszValue	Output parameter. Returns the image length, in bytes.

Discussion

If the item exists, this function returns the image and image length in the ppImage and pszValue parameters and returns M1_OK.

If the item does not exist, or if it exists but is not stored as an image, this function returns M1_ITEMNOTEXIST. It is the responsibility of the caller to know the general type of the item's data.

The returned image now belongs to the caller, who must call the M1Free function to release it.

Return Codes

M1_OK	Indicates that the operation was completed successfully.
M1_INVALIDPARAM	Indicates that a pointer parameter was null.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

M1Free

 Copyright © 2010 ImageWare Systems, Inc.

M1Free

Frees a generic OpenM1 memory block. Currently only to be used on memory allocated by the M1GetItemImage function.

Prototype

```
int M1Free(
     BYTE* pImage
);
```

Parameters

pImage	Input parameter. Specifies a pointer to a valid image.

Discussion

This function should be called after the M1GetItemImage function to free the memory buffer from that function.

Return Codes

M1_OK	Indicates that the operation was completed successfully.
M1_INVALIDPARAM	Indicates that the pointer parameter was null.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

M1GetItemImage

 Copyright © 2010 ImageWare Systems, Inc.

4.3 Validation and Serialization

The real heart of the OpenM1 API is in its validation. During this phase, OpenM1 will try to match up the *sections* and *items* specified by the caller with the *sectiondefs* and *itemdefs* specified in the standard configuration.

4.3.1 Validation Pass 1: Structure Adjustment

The first pass of the OpenM1 validation process is the called the *structure adjustment pass*, where a copy of the user's data is made piece-by-piece to allow for careful correlation with the standard's structure:

1. Each container definition (**containerdef**) in the standard's definition tree is checked to see if it is reserved. Reserved containers are set by OpenM1 and not by the user, so if the container is reserved, the item or section is explicitly added (recursively) to the section copy, **sectionAdjusted**.

   Otherwise, all containers in the user data that have the same name are collected and placed in a separate list, **containersNamed**.
2. If containersNamed is empty, the user has not provided this container, and the code checks to see if there is a default value. If so, the container is added to sectionAdjusted, otherwise an error is flagged and the process ends. This allows the user to skip setting fields that already have appropriate default values in the standard definition.
3. containersNamed is then checked against the occurrence limitations defined in the associated containerdef and an error is flagged and the process ends if any conditions have been breached.
4. containersNamed is sorted by index, and the provided indexes are checked to make sure they are indeed sequential and start at 1. Otherwise, an error is flagged and the process ends.
5. If the container is an item, it is added it to sectionAdjusted.
6. If the container is a section, it is recursed back into, using this section and its associated section definition (sectiondef) as parameters to continue the process one level down

Once this structure adjustment pass is complete, sectionAdjusted contains a tree structure that has been validated against the standard's tree structure, into which missing and reserved containers have been inserted.

4.3.2 Validation Pass 2: Data Coercion

To complete the validation process, the data within the items must be validated according to the rules imposed by the associated item definitions (**itemdef**). This occurs in the second pass, that is called the *data coercion pass*. In this pass:

1. For each item in sectionAdjusted the datatype of the value provided by the caller is *coerced* into the datatype defined by its itemdef. This step is necessary because OpenM1 does not require the user to know the exact final datatype of each final element. To make

 Copyright © 2010 ImageWare Systems, Inc.

a case in point, the following four function calls will result in the same final setting for the item Moustache of datatype bit:

```
M1SetItemBit(pFeatureMask,    "Moustache", 1, true);
M1SetItemNumber(pFeatureMask, "Moustache", 1, 1);
M1SetItemString(pFeatureMask, "Moustache", 1, "1");
M1SetItemString(pFeatureMask, "Moustache", 1, "true");
```

If the data provided cannot be converted into the final datatype, an error is flagged and the validation process ends.

2. Finally the data is checked against the optionally provided min and max conditions. For strings and binary types min and max refer to the lengths of the data. Any error is flagged and the process ends at the first error.

4.3.3 Validation Pass 3: Macro Resolution

The third and final validation step is the *macro resolution pass*, where the length() and occurrences() macros are evaluated. The description of this process is beyond the scope of this document. Please refer to the source code for an in-depth understanding of this step.

The M1ValidatePackage function returns a highly detailed log of all actions taken, as well as a detailed error string outlining any validation failures which should greatly simplify correcting the invalid package.

4.3.4 Validation and Serialization Function

The OpenM1 API contains the following validation and serialization functions:

Function	Description
M1ValidatePackage	Validates a root section against a standard, returning a detailed log of the process and any potential errors.
M1WritePackage	Writes a validated disk package to a disk file.
M1ReadPackage	Reads a package from file, using a specified standard as a base structure defining the file.

 Copyright © 2010 ImageWare Systems, Inc.

M1ValidatePackage

Validates a root section against a standard, returning a detailed log of the process and any potential errors. Package validation is necessary before writing the package.

Prototype

```
int M1ValidatePackage(
     M1Section* pHeadSection,
     M1Standard* pStandard,
     TCHAR** pszLog,
     TCHAR** pszError
);
```

Parameters

pHeadSection	Input parameter. Specifies a pointer to the root section to be validated.
pStandard	Input parameter. Specifies a pointer to an XML standard configuration file.
pszLog	Output parameter. Returns a detailed log of the steps taken by this function.
pszError	Optional output parameter. If validation fails, returns an error string.

Discussion

If validation succeeds, this function returns M1_OK. If validation fails, it returns M1_FAIL and sets detailed information in the returned error string. In both cases, success or failure, a detailed log of the steps taken is returned, which can assist in diagnosing invalid packages.

Return Codes

M1_OK	Indicates that validation was completed successfully.
M1_FAIL	Indicates that validation failed. Check the returned error string for details.
M1_INVALIDPARAM	Indicates that a pointer parameter was null.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

None.

 Copyright © 2010 ImageWare Systems, Inc.

M1WritePackage

Writes a validated disk package to a disk file.

Prototype

```
int M1WritePackage(
     M1Section* pHeadSection,
     const TCHAR* szFile
);
```

Parameters

pHeadSection	Input parameter. Specifies a pointer to the root section to be written to disk.
szFile	Input parameter. Specifies the path to which the file should be written.

Discussion

Note that the package must be validated against a standard using the M1ValidatePackage function, otherwise this function returns M1_NOTVALIDATED.

Return Codes

M1_OK	Indicates that the file was successfully wirtten.
M1_NOTVALIDATED	Indicates that the package has not yet passed validation.
M1_INVALIDPARAM	Indicates that a pointer parameter was null.
M1_FILEERROR	Indicates a generic file error.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

M1ValidatePackage

 Copyright © 2010 ImageWare Systems, Inc.

M1ReadPackage

Reads a package from file, using a specified standard as a base structure defining the file.

Prototype

```
int M1ReadPackage(
     const TCHAR* szFile,
     M1Section** ppHeadSection,
     M1Standard* pStandard,
     TCHAR** pszError
);
```

Parameters

szFile	Input parameter. Specifies the path of the file to be read.
ppHeadSection	Output parameter. Specifies a pointer to the root section.
pStandard	Input parameter. Specifies a pointer to an XML standard configuration file.
pszError	Optional output parameter. If operation fails, returns an error string.

Discussion

If the file is successfully read in, the function sets the *ppHeadSection parameter and returns M1_OK. If the file is not successfully read in, the function sets a detailed error message in pszError and returns M1_FAIL.

The read-in package must eventually be freed with the M1FreePackage function.

Note that after reading in a package with this function, its overall structure has been validated against the provided standard, but its contents have not been fully validated. To perform full package validation, call the M1ValidatePackage function.

Return Codes

M1_OK	Indicates that the operation was completed successfully.
M1_FAIL	Indicates that the read operation failed.
M1_INVALIDPARAM	Indicates that a pointer parameter was null.
M1_EXCEPTION	Indicates that an unexpected error occurred.

See Also

M1FreePackage
M1ValidatePackage

 Copyright © 2010 ImageWare Systems, Inc.

5 Samples

OpenM1 provides 5 short C++ samples demonstrating the functionality of OpenM1.

These samples make use of the XML standard configuration files that can be found under the Standards folder.

 Copyright © 2010 ImageWare Systems, Inc.

5.1 OpenM1Sample1

OpenM1Sample1 outputs detailed information on a set of XML standard configuration files by traversing the containerdefs recursively.

Functions used:

M1ReadStandard, M1FreeStandard, M1GetHeadContainer, M1GetContainerInfo, M1GetItemDefInfo, M1EnumContainerReset, M1EnumContainerGetNext

Example output for one one the XML standard configuration files is presented here.

```
Standard "..\Standards\ISO 19794-4.xml":
SectionDef "ISO 19794-4":
....SectionDef "General Record Header":
........ItemDef "Format Identifier": fmt(6) min(3) max(3) def(FIR) R
........ItemDef "Version Number": fmt(6) min(3) max(3) def(010) R
........ItemDef "Record Length": fmt(4) def(length(ISO 19794-4)) R
........ItemDef "Capture Device ID": fmt(3)
........ItemDef "Number of Fingers": fmt(2) min(1) def(occurrences(Finger Image
Record)) R
........ItemDef "Scale Units": fmt(2)
........ItemDef "Horizontal Scan Resolution": fmt(3)
........ItemDef "Vertical Scan Resolution": fmt(3)
........ItemDef "Horizontal Image Resolution": fmt(3)
........ItemDef "Vertical Image Resolution": fmt(3)
........ItemDef "Pixel Depth": fmt(2) min(1) max(16)
........ItemDef "Image Compression Algorithm": fmt(2)
....SectionDef "Finger Image Record": maxocc(unbounded)
........SectionDef "Finger Image Header":
............ItemDef "Finger Image Record Length": fmt(4) def(length(Finger Image
Record)) R
............ItemDef "Finger Position": fmt(2)
............ItemDef "Count of Views": fmt(2) min(1) max(255)
............ItemDef "View Number": fmt(2) min(1) max(255)
............ItemDef "Finger Image Quality": fmt(2) min(0) max(100)
............ItemDef "Impression Type": fmt(2)
............ItemDef "Horizontal Line Length": fmt(3)
............ItemDef "Vertical Line Length": fmt(3)
........ItemDef "Finger Image Data": fmt(0)
```

 Copyright © 2010 ImageWare Systems, Inc.

5.2 OpenM1Sample2

OpenM1Sample2 creates an ISO 19794-5 Face Recognition package by setting a few mandatory text and number fields and providing a sample image.

Functions used in this sample:

`M1ReadStandard`, `M1FreeStandard`, `M1CreatePackage`, `M1ReturnSection`, `M1SetItemNumber`, `M1SetItemBit`, `M1SetItemImage`, `M1ValidatePackage`, `M1WritePackage`, `M1FreePackage`.

This is the output for this sample program, which is the log string returned by the `M1ValidatePackage` function.

```
Pass 1: Verifying and adjusting general tree structure...
Adding reserved section "Facial Record Header"
Adding reserved item "Format Identifier" with default value "FAC"
Adding reserved item "Version Number" with default value "010"
Adding reserved item "Length of Record" with default value "length(ISO 19794-5)"
Adding reserved item "Number of Facial Images" with default value
"occurrences(Facial Record Data)"
Checking occurrences for 1 container(s) "Facial Record Data": 1..unbounded
Checking indices for container "Facial Record Data", index 1 of count 1
Checking occurrences for 1 container(s) "Facial Information Block": 1..1
Checking indices for container "Facial Information Block", index 1 of count 1
Adding reserved item "Face Image Block Length" with default value "length(Facial
Record Data)"
Adding reserved item "Number of Feature Points" with default value
"occurrences(Feature Points)"
Checking occurrences for 1 container(s) "Gender": 1..1
Checking indices for container "Gender", index 1 of count 1
Checking occurrences for 1 container(s) "Eye Color": 1..1
Checking indices for container "Eye Color", index 1 of count 1
Checking occurrences for 1 container(s) "Hair Color": 1..1
Checking indices for container "Hair Color", index 1 of count 1
Checking occurrences for 1 container(s) "Feature Mask": 1..1
Checking indices for container "Feature Mask", index 1 of count 1
Checking occurrences for 1 container(s) "Features Specified": 1..1
Checking indices for container "Features Specified", index 1 of count 1
Checking occurrences for 1 container(s) "Glasses": 1..1
Checking indices for container "Glasses", index 1 of count 1
Checking occurrences for 1 container(s) "Moustache": 1..1
Checking indices for container "Moustache", index 1 of count 1
Adding missing item "Beard" with default value "0"
Checking occurrences for 1 container(s) "Beard": 1..1
Checking indices for container "Beard", index 1 of count 1
Adding missing item "Teeth Visible" with default value "0"
Checking occurrences for 1 container(s) "Teeth Visible": 1..1
Checking indices for container "Teeth Visible", index 1 of count 1
Adding missing item "Blink" with default value "0"
Checking occurrences for 1 container(s) "Blink": 1..1
Checking indices for container "Blink", index 1 of count 1
Adding missing item "Mouth Open" with default value "0"
Checking occurrences for 1 container(s) "Mouth Open": 1..1
Checking indices for container "Mouth Open", index 1 of count 1
Adding missing item "Left Eye Patch" with default value "0"
Checking occurrences for 1 container(s) "Left Eye Patch": 1..1
Checking indices for container "Left Eye Patch", index 1 of count 1
Adding missing item "Right Eye Patch" with default value "0"
```

 Copyright © 2010 ImageWare Systems, Inc.

```
Checking occurrences for 1 container(s) "Right Eye Patch": 1..1
Checking indices for container "Right Eye Patch", index 1 of count 1
Adding missing item "Dark Glasses" with default value "0"
Checking occurrences for 1 container(s) "Dark Glasses": 1..1
Checking indices for container "Dark Glasses", index 1 of count 1
Adding missing item "Feature Distorting Medical Condition" with default value "0"
Checking occurrences for 1 container(s) "Feature Distorting Medical Condition":
1..1
Checking indices for container "Feature Distorting Medical Condition", index 1 of
count 1
Adding reserved item "Feature Mask bit 11" with default value "0"
Adding reserved item "Feature Mask bit 12" with default value "0"
Adding reserved item "Feature Mask bit 13" with default value "0"
Adding reserved item "Feature Mask bit 14" with default value "0"
Adding reserved item "Feature Mask bit 15" with default value "0"
Adding reserved item "Feature Mask bit 16" with default value "0"
Adding reserved item "Feature Mask bit 17" with default value "0"
Adding reserved item "Feature Mask bit 18" with default value "0"
Adding reserved item "Feature Mask bit 19" with default value "0"
Adding reserved item "Feature Mask bit 20" with default value "0"
Adding reserved item "Feature Mask bit 21" with default value "0"
Adding reserved item "Feature Mask bit 22" with default value "0"
Adding reserved item "Feature Mask bit 23" with default value "0"
Checking for extraneous containers in section "Feature Mask"
Checking occurrences for 1 container(s) "Expression": 1..1
Checking indices for container "Expression", index 1 of count 1
Adding missing item "Pose Angle - Yaw" with default value "0"
Checking occurrences for 1 container(s) "Pose Angle - Yaw": 1..1
Checking indices for container "Pose Angle - Yaw", index 1 of count 1
Adding missing item "Pose Angle - Pitch" with default value "0"
Checking occurrences for 1 container(s) "Pose Angle - Pitch": 1..1
Checking indices for container "Pose Angle - Pitch", index 1 of count 1
Adding missing item "Pose Angle - Roll" with default value "0"
Checking occurrences for 1 container(s) "Pose Angle - Roll": 1..1
Checking indices for container "Pose Angle - Roll", index 1 of count 1
Adding missing item "Pose Angle Uncertainty - Yaw" with default value "0"
Checking occurrences for 1 container(s) "Pose Angle Uncertainty - Yaw": 1..1
Checking indices for container "Pose Angle Uncertainty - Yaw", index 1 of count 1
Adding missing item "Pose Angle Uncertainty - Pitch" with default value "0"
Checking occurrences for 1 container(s) "Pose Angle Uncertainty - Pitch": 1..1
Checking indices for container "Pose Angle Uncertainty - Pitch", index 1 of count
1
Adding missing item "Pose Angle Uncertainty - Roll" with default value "0"
Checking occurrences for 1 container(s) "Pose Angle Uncertainty - Roll": 1..1
Checking indices for container "Pose Angle Uncertainty - Roll", index 1 of count 1
Checking for extraneous containers in section "Facial Information Block"
Checking occurrences for 2 container(s) "Feature Points": 0..unbounded
Checking indices for container "Feature Points", index 1 of count 2
Checking indices for container "Feature Points", index 2 of count 2
Adding reserved item "Feature Type" with default value "1"
Checking occurrences for 1 container(s) "Feature Point": 1..1
Checking indices for container "Feature Point", index 1 of count 1
Checking occurrences for 1 container(s) "Horizontal Position": 1..1
Checking indices for container "Horizontal Position", index 1 of count 1
Checking occurrences for 1 container(s) "Vertical Position": 1..1
Checking indices for container "Vertical Position", index 1 of count 1
Adding reserved item "Reserved" with default value "0"
Checking for extraneous containers in section "Feature Points"
Adding reserved item "Feature Type" with default value "1"
Checking occurrences for 1 container(s) "Feature Point": 1..1
Checking indices for container "Feature Point", index 1 of count 1
Checking occurrences for 1 container(s) "Horizontal Position": 1..1
```

 Copyright © 2010 ImageWare Systems, Inc.

```
Checking indices for container "Horizontal Position", index 1 of count 1
Checking occurrences for 1 container(s) "Vertical Position": 1..1
Checking indices for container "Vertical Position", index 1 of count 1
Adding reserved item "Reserved" with default value "0"
Checking for extraneous containers in section "Feature Points"
Checking occurrences for 1 container(s) "Image Information": 1..1
Checking indices for container "Image Information", index 1 of count 1
Checking occurrences for 1 container(s) "Face Image Type": 1..1
Checking indices for container "Face Image Type", index 1 of count 1
Adding reserved item "Image Data Type" with default value ""
Checking occurrences for 1 container(s) "Width": 1..1
Checking indices for container "Width", index 1 of count 1
Checking occurrences for 1 container(s) "Height": 1..1
Checking indices for container "Height", index 1 of count 1
Checking occurrences for 1 container(s) "Image Color Space": 1..1
Checking indices for container "Image Color Space", index 1 of count 1
Checking occurrences for 1 container(s) "Source Type": 1..1
Checking indices for container "Source Type", index 1 of count 1
Adding missing item "Device Type" with default value "0"
Checking occurrences for 1 container(s) "Device Type": 1..1
Checking indices for container "Device Type", index 1 of count 1
Adding reserved item "Quality" with default value "0"
Checking for extraneous containers in section "Image Information"
Checking occurrences for 1 container(s) "Image Data": 1..1
Checking indices for container "Image Data", index 1 of count 1
Checking for extraneous containers in section "Facial Record Data"
Checking for extraneous containers in section "ISO 19794-5"
Pass 2: Validating individual items...
Validating item "Format Identifier"
Validating item "Version Number"
Validating item "Length of Record"
Validating item "Number of Facial Images"
Validating item "Face Image Block Length"
Validating item "Number of Feature Points"
Validating item "Gender"
Verified legal name/value "Male"/"1" for item Gender
Validating item "Eye Color"
Verified legal name/value "Gray"/"4" for item Eye Color
Validating item "Hair Color"
Verified legal name/value "Brown"/"4" for item Hair Color
Validating item "Features Specified"
Validating item "Glasses"
Validating item "Moustache"
Validating item "Beard"
Validating item "Teeth Visible"
Validating item "Blink"
Validating item "Mouth Open"
Validating item "Left Eye Patch"
Validating item "Right Eye Patch"
Validating item "Dark Glasses"
Validating item "Feature Distorting Medical Condition"
Validating item "Feature Mask bit 11"
Validating item "Feature Mask bit 12"
Validating item "Feature Mask bit 13"
Validating item "Feature Mask bit 14"
Validating item "Feature Mask bit 15"
Validating item "Feature Mask bit 16"
Validating item "Feature Mask bit 17"
Validating item "Feature Mask bit 18"
Validating item "Feature Mask bit 19"
Validating item "Feature Mask bit 20"
Validating item "Feature Mask bit 21"
```

```
Validating item "Feature Mask bit 22"
Validating item "Feature Mask bit 23"
Validating item "Expression"
Verified legal name/value "Neutral"/"1" for item Expression
Validating item "Pose Angle - Yaw"
Validating item "Pose Angle - Pitch"
Validating item "Pose Angle - Roll"
Validating item "Pose Angle Uncertainty - Yaw"
Validating item "Pose Angle Uncertainty - Pitch"
Validating item "Pose Angle Uncertainty - Roll"
Validating item "Feature Type"
Validating item "Feature Point"
Validating item "Horizontal Position"
Validating item "Vertical Position"
Validating item "Reserved"
Validating item "Feature Type"
Validating item "Feature Point"
Validating item "Horizontal Position"
Validating item "Vertical Position"
Validating item "Reserved"
Validating item "Face Image Type"
Verified legal name/value "Basic"/"1" for item Face Image Type
Validating item "Image Data Type"
Verified legal name/value "JPEG"/"0" for item Image Data Type
Validating item "Width"
Validating item "Height"
Validating item "Image Color Space"
Verified legal name/value "24 bit RGB"/"1" for item Image Color Space
Validating item "Source Type"
Verified legal name/value "Static photograph from a digital still-image
camera"/"2" for item Source Type
Validating item "Device Type"
Validating item "Quality"
Validating item "Image Data"
Pass 3: Resolving macros...
Resolving macro "length(ISO 19794-5)" = "length" of "ISO 19794-5"
Found container "ISO 19794-5" of byte length 60905
Resolving macro "occurrences(Facial Record Data)" = "occurrences" of "Facial
Record Data"
Found 1 occurrences of container "Facial Record Data"
Resolving macro "length(Facial Record Data)" = "length" of "Facial Record Data"
Found container "Facial Record Data" of byte length 60891
Resolving macro "occurrences(Feature Points)" = "occurrences" of "Feature Points"
Found 2 occurrences of container "Feature Points"
```

 Copyright © 2010 ImageWare Systems, Inc.

5.3 OpenM1Sample3

OpenM1Sample3 reads in an ISO 19794-5 Face Recognition package and outputs a text field, a number field, and extracts the face image and writes it to file.

Functions used in this sample:

`M1ReadStandard`, `M1FreeStandard`, `M1ReadPackage`, `M1ReturnSection`, `M1GetItemString`, `M1GetItemNumber`, `M1GetItemImage`, `M1FreePackage`, `M1Free`.

This sample opens the package binary and using the `M1Get*` functions outputs:

```
Item "Format Identifier" has value "FAC".
Item "Number of Facial Images" has value "1".
```

and writes the extracted image to `sample.jpg`.

5.4 OpenM1Sample4

OpenM1Sample4 creates an ISO 19794-4 Fingerprint Image package by setting a few mandatory text and number fields and providing a sample image.

Functions used in this sample:

`M1ReadStandard`, `M1FreeStandard`, `M1CreatePackage`, `M1ReturnSection`, `M1SetItemNumber`, `M1SetItemImage`, `M1ValidatePackage`, `M1WritePackage`, `M1FreePackage`.

5.5 OpenM1Sample5

OpenM1Sample5 creates an ISO 19794-4 Fingerprint Image package by setting a few mandatory text and number fields and providing a sample image.

Functions used in this sample:

`M1ReadStandard`, `M1FreeStandard`, `M1CreatePackage`, `M1ReturnSection`, `M1SetItemNumber`, `M1SetItemImage`, `M1ValidatePackage`, `M1WritePackage`, `M1FreePackage`.

 Copyright © 2010 ImageWare Systems, Inc.

INDEX

C++ samples.....
OpenM1Sample1.....39
OpenM1Sample2.....40
OpenM1Sample3.....44
OpenM1Sample4.....44
OpenM1Sample5.....44
configuration file.....7
container.....6
containerdef.....6
functions.....
data access.....19
serialization.....34
standard configuration.....9
validation.....34
item.....6
itemdef.....6
M1Container.....6
M1ContainerDef.....7
M1CreatePackage function.....**20**
M1EnumContainerGetNext function.....**16**
M1EnumContainerReset function.....**15**
M1Free function.....**32**
M1FreePackage function.....**21**
M1FreeStandard function.....**11**
M1GetContainerInfo function.....**13**
M1GetHeadContainer function.....**12**
M1GetItemDefInfo function.....**17**
M1GetItemImage function.....**31**
M1GetItemNumber function.....**30**
M1GetItemString function.....**29**
M1Item.....6
M1ItemDef.....7
M1ReadPackage function.....**37**
M1ReadStandard function.....**10**
M1ReturnSection function.....**22**
M1Section.....6
M1SectionDef.....7
M1SetItemBit function.....**26**
M1SetItemImage function.....**27**
M1SetItemNumber function.....**25**
M1SetItemString function.....**24**
M1ValidatePackage function.....**35**
M1WritePackage function.....**36**
recursion.....6
samples.....
OpenM1Sample1.....39
OpenM1Sample2.....40
OpenM1Sample3.....44
OpenM1Sample4.....44
OpenM1Sample5.....44
section.....6
sectiondef.....6
serialization functions.....34
standard configuration file.....7
standard configuration functions.....9
validation functions.....34

 Copyright © 2010 ImageWare Systems, Inc.