SYSTEM 7000 External Membership Interface

Stored Procedure Interface

Interface Specification

Version: 2.0
Date Created: 24-Apr-09
Date Last Updated: 27-Apr-16
Last Updated By: Surekha Ramisetty

Authorisation

<table>
<thead>
<tr>
<th>By</th>
<th>Name</th>
<th>Position</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Originator</td>
<td>Surekha Ramisetty</td>
<td>Team Lead - Applications</td>
<td></td>
</tr>
<tr>
<td>Approver</td>
<td>Sen Ly</td>
<td>Systems R&amp;D Architect</td>
<td></td>
</tr>
<tr>
<td>Approver</td>
<td>Paul Soo</td>
<td>Test and Release Manager</td>
<td></td>
</tr>
<tr>
<td>Approver</td>
<td>David Jack</td>
<td>Product Manager</td>
<td></td>
</tr>
<tr>
<td>Approver</td>
<td>Robert Taranto</td>
<td>Manager R&amp;D Systems and Networks</td>
<td></td>
</tr>
</tbody>
</table>
6 GET NEXT MEMBERSHIP NUMBER ........................................................................................................18

6.1 Function GET_NEXT_MEMBERNUMBER ..................................................................................18

6.1.1 Calling the function and checking the result ........................................................................18

7 C# EXAMPLE ..................................................................................................................................19

8 VB.NET EXAMPLE .........................................................................................................................25

9 VISUAL BASIC EXAMPLE ..........................................................................................................31

10 DOCUMENT CHANGE HISTORY ..............................................................................................34
1 Preface

1.1 Purpose

This Interface Specification describes a standard method, proprietary to Aristocrat, whereby a venue’s External Membership System can transfer membership information into System 7000 PRIME by means of calling stored procedures provided by Aristocrat.

The stored procedures provided allow the membership system to update S7000 with:

- new members
- modification to existing members
- members to be deleted or removed
- changes to a member’s membership card

1.2 Scope

This document is intended for the software development personnel of the Membership System that intends to interface to System 7000.

It is assumed that the reader of this document has an understanding of relational databases and is familiar in the use of stored procedure calls to implement a client-server type application. The interface described in this document is only available on versions of System 7000 that have had the relevant database upgrade.

Aristocrat reserves the right to change this interface and the information in this document without notice.

The Membership System is fully responsible for the development, maintenance and use of their interface to S7000. The Membership System is required to ensure the developed interface is working prior to installation at an venue, and is used in accordance to any regulatory requirements that applies.

Aristocrat does not assume responsibility for errors or for any consequences arising from errors or losses in the provision or use of this interface.

This document contains intellectual property of Aristocrat and contents must not be disclosed to parties not having a non-disclosure agreement with Aristocrat.

All coding examples shown in this document as provided as a guide to assist in development of the interface to S7000. These examples might not be programmatically correct.

The Membership System is not to implement this interface at a S7000 venue without the prior consent of Aristocrat. The username and the password required to connect to S7000 should be obtained and enabled from Aristocrat prior to the development of the interface.
2 Overview

2.1 Membership System Responsibility

The Membership System is responsible for managing the venue’s members and issuing membership cards to the members. All new members are registered into this system and changes to a member's details are done on this system.

Member and card additions and changes that affect the access of the members to S7000 promotional facilities are updated into S7000 by calling the stored procedure/s provided.

Membership system is responsible for implementing their interface to our system, i.e. developing the interface which calls to our stored procedures detailed in this document, and ensuring the correct actions are initiated in S7000.

The membership system is considered to be the true source of membership information in the venue.

It is the responsibility of the Membership system to update S7000 in a timely manner. New members or cards will not be recognised by S7000 until updated by the Membership system.

High volume membership changes should be scheduled to be updated into S7000 in periods of low activity.

2.2 Security

The described stored procedures cannot be used by default. It has to be enabled by Aristocrat Support.

The username and the password required to connect to S7000 should be obtained from Aristocrat prior to the development of the interface. This information must be secured by the Membership System and not be disclosed to unauthorised parties.

2.3 Legal Requirements

The Membership System is responsible for complying with any regulatory requirements stemming from the use of this interface at a venue.

2.4 Interfaces

Membership data updates into S7000 is by means of a stored procedure interface created by the membership system to call the stored procedures provided.

2.5 Hardware, Software and Network Requirements

The interface described in this document is only available on S7000 version 7.3 and later.

2.6 Rules

The membership system is the master of all member details and cards. It is responsible for sending the correct details to S7000. S7000 will consider any data passed to it, via this interface, as current.
Membership system will have to send all the required details each time. Member details are updated in S7000 if the member is defined; else a new member is created. Corresponding values in S7000 will be overwritten when updating.

S7000 has to be updated with the membership changes if the changes affect the access of the members to the S7000 promotional facilities.

- New members and cards have to be added to S7000.
- Expired or lost cards and ex-members have to be removed from S7000.
- Members no longer allowed access to S7000 promotional facilities should have their cards removed from S7000.

Any member notified to S7000 for deletion will be deleted in S7000, including all history and promotional points/rewards stored within S7000.

This interface will only support the transfer of membership details into S7000. Any changes to members’ details and cards within S7000 will not be notified back to the membership system, and could be overwritten on a subsequent update from the membership system.

The membership system has to ensure that the member details associated with a Member Number is not changed to another member’s without first deleting the original Member Number. Not doing so will result in the new person inheriting any remaining promotional points and play history of the original person.

The following limitations apply on the data being updated into S7000:

- Card numbers must be 10 digits
- Only 1 card will be permitted per member, only that supplied by the Membership system.
  - If supplied card number is not defined in S7000, that card is allocated.
  - All other cards will be deleted.
  - If no card number is supplied, all cards associated to that member are deleted in S7000
- Barcode For Mail Address
  - System 7000 supports only the “Rapid Addressing Tool” barcode format.
  - System 7000 will not check if the supplied barcode is a valid, and is recorded as supplied.

### 2.7 Glossary

The following terms, acronyms and abbreviations are used in this document.

<table>
<thead>
<tr>
<th>Term/Acronym/Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>S7000</td>
<td>Aristocrat’s System 7000 PRIME casino management system</td>
</tr>
<tr>
<td>EGM</td>
<td>Electronic Gaming Machine, a.k.a. a slot machine.</td>
</tr>
<tr>
<td>Member</td>
<td>A person who has joined the venue and is required to be identified by S7000.</td>
</tr>
<tr>
<td>Card</td>
<td>A magnetic strip card, encoded with a unique number, used to identify the member to S7000 when used at a S7000 promotional device</td>
</tr>
</tbody>
</table>
3 USING THE INTERFACE

Membership system is required to:

- make a connection into the S7000 Oracle database using the login details provided.
- make the stored procedure call/s on the S7000 database to update membership identification data. One stored procedure call per member.
- handle and recover from any errors or exceptions generated on connecting to the S7000 database or when calling the stored procedure call.
- close the database connection on completion.

All the member information required to be updated in S7000 system is contained within the stored procedure call itself as part of the parameters/arguments list and a single returned value indicating the success or failure of the procedure.

3.1 Connection

For this interface, the relationship between the external Membership system and System 7000 PRIME will be a client-server type relationship, with the System 7000 PRIME Host being the database server and a computer on the external system being the client.

The client is required to make a connection into the System 7000 PRIME database. The client will have to make stored procedure calls on the System 7000 PRIME Oracle database to add, update or delete members and/or member cards.

The connection between the System 7000 PRIME and the external systems will be by means of a TCP/IP network connection between the System 7000 PRIME Host computer and a computer on the external system.

For Windows based clients, an ODBC database connection is recommended.

3.2 Client Software

“Oracle 10g R2 Client Software” has to be installed on the client which requires a connection to the System 7000 PRIME database.

This software will provide

- the Oracle Net software necessary to connect to the Oracle Database Server and
- the Oracle ODBC Driver software.

After installing the software the tnsnames.ora file should be altered to point to the System 7000 PRIME Oracle Database Server. This file can be found in the ORACLE_HOME\network\admin folder.
An entry should be made in this file similar to the following:

```plaintext
S7S =
  (DESCRIPTION =
   (ADDRESS_LIST =
    (ADDRESS =
      (PROTOCOL = TCP) (HOST = 172.10.36.25) (PORT = 1521)
    )
   )
   (CONNECT_DATA = (SERVICE_NAME = S7S)
    )
  )
```

where the Host IP address is the IP address of the Oracle database server. The SERVICE_NAME will always be S7S for a System 7000 PRIME database system.

### 3.3 Setting up an ODBC Data Source

This step is required only if using the ODBC connection to communicate with the System 7000 PRIME Database Server. The step can be skipped if the Oracle Client Software is to be used.

Once the Oracle Client Software has been installed, a System Data Source should be set up to the System 7000 PRIME Oracle Database Server using:

1. The Data Sources (ODBC) Applet within the Control Panel > Administrative Tools menu of Windows.
2. Select Add.
3. The Create New Data Source screen will be displayed.
4. Choose Oracle ODBC Driver and select Finish.

5. The Oracle ODBC Driver Configuration screen will be displayed.

6. Enter a Data Source name and description.

7. Enter the TNS Service Name that was entered into the tnsnames.ora file. In this example: S7S.

8. Enter the User ID for the database connection. Note: Use the User ID supplied by Aristocrat appropriate for this interface.

9. Select OK. (If required the connection to the database can be tested by selecting Test Connection button)

10. The System Data Source has been configured.

3.4 Database Connection

The following username and password can be used in the interface to connect into the System 7000 PRIME database:

User Name = MEMBCON
Password = MEMBCON

However this username and password should be validated and enabled from Aristocrat prior to the development of the interface.
4 MEMBER ADD OR UPDATE

4.1 Stored Procedure SP_ADD_UPDATE_MEMBER

The Membership System may call the stored procedure SP_ADD_UPDATE_MEMBER for the following operations:

1. Add a new member into S7000.

2. Update or replace existing member’s details.

3. Allocate new active card number or activate an existing card for a member.

4. Delete cards associated with a member.

When this stored procedure is called, the member details are updated in S7000 if member is present; else a new member will be created.

Following is the parameter structure of the stored procedure:

```
SP_ADD_UPDATE_MEMBER
(
  p_member_number IN VARCHAR2, --member number
  p_first_name IN VARCHAR2,  --member first name
  p_surname IN VARCHAR2,    --member surname
  p_title IN VARCHAR2,       --member title
  p_date_of_birth IN DATE,   --member birthday
  p_join_date IN DATE,       --member join/register date
  p_gender IN VARCHAR2,      --member gender
  p_telephone IN VARCHAR2,   --member telephone
  p_mobile_phone IN VARCHAR2,--member mobile phone
  p_address_line1 IN VARCHAR2,--member 1st address
  p_address_line2 IN VARCHAR2,--member 2nd address
  p_address_line3 IN VARCHAR2,--member 3rd address
  p_address_line4 IN VARCHAR2,--member 4th address
  p_post_code IN VARCHAR2,   --member post code
  p_mail_barcode IN VARCHAR2,--member barcode
  p_email IN VARCHAR2,       --member email address
  p_card_number IN VARCHAR2,--member card no.
  p_card_expiry_date IN DATE,--card expiry date
  p_suspension_status IN NUMBER,--member suspension flag
  o_return_code OUT NUMBER,     --Output return code
  o_return_msg OUT VARCHAR2,   --Output return message
  o_mem_status IN NUMBER default 0,--member status
)
```

4.1.1 Calling the stored procedure

The following table describes the input and output parameters of SP_ADD_UPDATE_MEMBER stored procedure.

The Membership System when calling the stored is required to supply the details for all the input parameters. The stored procedure will return an output parameter which indicates the result, success or the failure of the member add/update process.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN/OUT</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_member_number</td>
<td>IN</td>
<td>VARCHAR2(12)</td>
<td>Unique identifier for the member, e.g. the membership number. All information regarding the member is associated with this Member Number. This parameter can not be null.</td>
</tr>
<tr>
<td>p_firstname</td>
<td>IN</td>
<td>VARCHAR2(25)</td>
<td>First name of member. This parameter can not be null.</td>
</tr>
<tr>
<td>p_surname</td>
<td>IN</td>
<td>VARCHAR2(25)</td>
<td>Surname of member. This parameter can not be null.</td>
</tr>
<tr>
<td>p_title</td>
<td>IN</td>
<td>VARCHAR2(25)</td>
<td>Title of member, e.g. Mr., etc.</td>
</tr>
<tr>
<td>p_date_of_birth</td>
<td>IN</td>
<td>DATE</td>
<td>Date of birth of member. This parameter can not be null.</td>
</tr>
<tr>
<td>p_join_date</td>
<td>IN</td>
<td>DATE</td>
<td>Date when member first joined or registered. If not supplied, this will be defaulted to current system date.</td>
</tr>
<tr>
<td>p_gender</td>
<td>IN</td>
<td>VARCHAR2(1)</td>
<td>'M' or 'F' for the member’s gender. 'M' – for Male 'F' – for Female If not supplied the gender will be set as Unknown in S7000.</td>
</tr>
<tr>
<td>p_telephone</td>
<td>IN</td>
<td>VARCHAR2(20)</td>
<td>Home or primary telephone number to go to the main number</td>
</tr>
<tr>
<td>p_mobile_phone</td>
<td>IN</td>
<td>VARCHAR2(20)</td>
<td>Mobile phone number.</td>
</tr>
<tr>
<td>p_address_line1</td>
<td>IN</td>
<td>VARCHAR2(60)</td>
<td>Street address of member (line 1).</td>
</tr>
<tr>
<td>p_address_line2</td>
<td>IN</td>
<td>VARCHAR2(60)</td>
<td>Street address of member (line 2).</td>
</tr>
<tr>
<td>p_address_line3</td>
<td>IN</td>
<td>VARCHAR2(60)</td>
<td>Street address of member (line 3).</td>
</tr>
<tr>
<td>p_address_line4</td>
<td>IN</td>
<td>VARCHAR2(60)</td>
<td>Street address of member (line 4).</td>
</tr>
<tr>
<td>p_post_code</td>
<td>IN</td>
<td>VARCHAR2(12)</td>
<td>Postal code for member.</td>
</tr>
<tr>
<td>p_mail_barcode</td>
<td>IN</td>
<td>VARCHAR2(37)</td>
<td>Mail Barcode for the above address.</td>
</tr>
<tr>
<td>p_email</td>
<td>IN</td>
<td>VARCHAR2(64)</td>
<td>Email address.</td>
</tr>
<tr>
<td>p_card_number</td>
<td>IN</td>
<td>VARCHAR2(10)</td>
<td>10 digit membership card number. If the card number is not supplied, the member will be created without a card, however such a member will not be recognised on an EGM.</td>
</tr>
<tr>
<td>p_card_expiry_date</td>
<td>IN</td>
<td>DATE</td>
<td>Expiry date of the supplied card. If the card expiry date is not supplied, the expiry date will be defaulted to the expiry date based on the Card Type Expiry Date configuration setting of S7000 Administration module.</td>
</tr>
<tr>
<td>p_suspension_status</td>
<td>IN</td>
<td>NUMBER(1)</td>
<td>Indicates the current suspension status of the member. 0 = member is not suspended, else suspended. This parameter can not be null.</td>
</tr>
<tr>
<td>o_return_code</td>
<td>OUT</td>
<td>NUMBER(5)</td>
<td>Output parameter from the stored procedure i.e. the return code which indicates the result.</td>
</tr>
<tr>
<td>Parameter</td>
<td>IN/OUT</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
|             |        |         | **o_return_str**  
| OUT         | VARCHAR2 (255)  
|             | Output parameter from the stored procedure i.e. the string that gives more information about the above return code; it indicates the result of the procedure whether the member add/update process was successful or has failed as a result of incorrect data supplied or for any other reason.  
|             | Refer to section 5.1.2 for detailed information.                                                                 |
|             |        |         | **p_mem_status**  
| IN          | NUMBER  | The number which represent member’s current status of membership with the club. The values supported are:  
|             |        | 0 – for **Pending** status  
|             |        | 1 – for **Active** or **Financial** status  
|             |        | 2 – for **Un-financial** status  
|             |        | 3 – for **Retired** status  
|             |        | If this parameter is not supplied the member status will be determined based on the card expiry date. If supplied the member status will be updated in S7000 with the supplied value.                                                                 |

4.1.2 Checking the result of the procedure

The output code and the string returned by the stored procedure sp_add_update_member will be one of the following:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Success</td>
</tr>
<tr>
<td>0</td>
<td>Failed. Contains the exception generated by the stored procedure</td>
</tr>
<tr>
<td>-1</td>
<td>The supplied Member Number is not valid</td>
</tr>
<tr>
<td>-2</td>
<td>Member First name is missing</td>
</tr>
<tr>
<td>-3</td>
<td>Member Surname is missing</td>
</tr>
<tr>
<td>-4</td>
<td>Member Date of Birth is missing</td>
</tr>
<tr>
<td>-5</td>
<td>The value supplied for gender is not valid</td>
</tr>
<tr>
<td>-6</td>
<td>The supplied Card Number is not valid</td>
</tr>
<tr>
<td>-7</td>
<td>The Suspension Status is missing</td>
</tr>
<tr>
<td>-8</td>
<td>Exception when inserting member. Contains the exception generated.</td>
</tr>
<tr>
<td>-9</td>
<td>Exception when updating member. Contains the exception generated.</td>
</tr>
<tr>
<td>-10</td>
<td>Supplied Card Number belongs to a different member</td>
</tr>
<tr>
<td>-11</td>
<td>Exception when adding card. Contains the exception generated.</td>
</tr>
<tr>
<td>-12</td>
<td>Exception when updating card. Contains the exception generated.</td>
</tr>
<tr>
<td>-13</td>
<td>Exception when updating Member Other information step failed. Contains the exception generated.</td>
</tr>
</tbody>
</table>
### 4.1.3 Add a new Member

When adding a new member the Membership system should send all the above details each time.

If a member already exists with the same Member Number as supplied, the S7000 system will just update the member details; else a new member with the supplied Member Number and details will be created.

If a Card Number is supplied, the card is allocated to the member.

If a card with the same Card Number as supplied already exists in S7000 and it is allocated to some other member, the stored procedure can not complete the member add process successfully.

### 4.1.4 Update or replace existing member’s details.

Membership system is required to send all the above details each time even when updating a member. Corresponding values in S7000 will be overwritten by the update.

Member details are only updated if member is present; else a new member will be created.

When updating a member if the supplied card number is not present in S7000, the card is allocated, and all other cards associated to that member will be deleted.

If no card number is supplied, all cards associated to that member are deleted in S7000.

If the supplied card number already exists in S7000 and is allocated to a different member, the stored procedure can not complete the member update process successfully.

### 4.1.5 Allocate new active card number or activate an existing card for a member.

Membership system is required to send all the above details each time even when intending to allocate a new card or to activate existing card for a member.

In order to allocate a new card or activate existing card, the Membership System has to supply a card number and the card expiry along with the other details when calling the stored procedure.

Only one card will be permitted per member, only that supplied by the Membership System when adding or updating member.

If the supplied card number is not present in S7000, the card is allocated, and all other cards will be deleted. If no card number is supplied, all cards associated to that member are deleted in S7000.

### 4.1.6 Delete cards associated with a member.

In order to delete cards associated with a member, the Membership system is required to send all the above details except the card number and card expiry date.

If the card number is not supplied for any member when calling the stored procedure, all cards associated to that member are automatically deleted in S7000.
4.2 Stored Procedure SP_ADD_UPDATE_MEMBER_EXT

The stored procedure SP_ADD_UPDATE_MEMBER_EXT as the name implies is an extension of the SP_ADD_UPDATE_MEMBER stored procedure described in section 4.1. The extension stored procedure allows supplying Member analysis type and values in addition to the basic details described in the earlier section.

Therefore the Membership System may call the stored procedure sp_add_update_member_ext for the following operations:

1. Add a new member into S7000.
2. Update or replace existing member’s details.
3. Allocate new active card number or activate an existing card for a member.
4. Delete cards associated with a member.
5. Replace existing member’s analysis details

When this stored procedure is called, the member details are updated in S7000 if member is present; else a new member will be created.

All the rules described in the section 4.1 regarding adding and updating of member and cards are also applicable to the extension stored procedure sp_add_update_member_ext.

Following is the parameter structure of the stored procedure:

```sql
SP_ADD_UPDATE_MEMBER_EXT
{
    p_member_number    IN VARCHAR2,  --member number
    p_firstname        IN VARCHAR2,  --member first name
    p_surname          IN VARCHAR2,  --member surname
    p_title            IN VARCHAR2,  --member title
    p_date_of_birth    IN DATE,      --member birthday
    p_join_date        IN DATE,      --member join/register date
    p_gender           IN VARCHAR2,  --member gender
    p_telephone        IN VARCHAR2,  --member telephone
    p_mobile_phone     IN VARCHAR2,  --member mobile phone
    p_address_line1    IN VARCHAR2,  --member 1st address
    p_address_line2    IN VARCHAR2,  --member 2nd address
    p_address_line3    IN VARCHAR2,  --member 3rd address
    p_address_line4    IN VARCHAR2,  --member 4th address
    p_post_code        IN VARCHAR2,  --member post code
    p_mail_barcode     IN VARCHAR2,  --member barcode
    p_email            IN VARCHAR2,  --member email address
    p_card_number      IN VARCHAR2,  --member card no.
    p_card_expiry_date IN DATE,      --card expiring date
    p Suspension status IN NUMBER,    --member suspension flag
    p_analysis_array   IN twodm_sa_array_t, --member analysis type and value array
    o_return_code      OUT NUMBER,    --Output return code
    o_return_str       OUT VARCHAR2,  --Output return message
    p_mem_status       IN NUMBER default 0  --member status
}
```

4.2.1 Calling the stored procedure

In addition to the input and output parameters described in the section 4.1.1 the Membership System is required to supply an array of analysis type and values as an additional parameter when
calling the extension stored procedure SP_ADD_UPDATE_MEMBER_EXT. The additional parameter is as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN/OUT</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_analysis_array</td>
<td>IN</td>
<td>Two-dimensional user defined string array</td>
<td>Array of Analysis Type and value as member analysis. The analysis type and value should be supplied as a pair. This parameter accepts the numeric Analysis Type Code and the Analysis Value Code as configured in the S7000 system. E.g.: p_analysis_array(1)(1) := 02; p_analysis_array(1)(2) := 5; p_analysis_array(2)(1) := 02; p_analysis_array(2)(2) := 10; p_analysis_array(3)(1) := 05; p_analysis_array(3)(2) := 25;</td>
</tr>
</tbody>
</table>

If this parameter is supplied, the existing member analysis details will be replaced with the supplied details.
If this parameter is not supplied, the existing member analysis details will be deleted in the S7000. Hence the membership system should supply full details each time.

### 4.2.2 Checking the result of the procedure

The output code and the string returned by the stored procedure sp_add_update_member_ext will be one of the following:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Success</td>
</tr>
<tr>
<td>0</td>
<td>Failed. Contains the exception generated by the stored procedure</td>
</tr>
<tr>
<td>-1</td>
<td>The supplied Member Number is not valid</td>
</tr>
<tr>
<td>-2</td>
<td>Member First name is missing</td>
</tr>
<tr>
<td>-3</td>
<td>Member Surname is missing</td>
</tr>
<tr>
<td>-4</td>
<td>Member Date of Birth is missing</td>
</tr>
<tr>
<td>-5</td>
<td>The value supplied for gender is not valid</td>
</tr>
<tr>
<td>-6</td>
<td>The supplied Card Number is not valid</td>
</tr>
<tr>
<td>-7</td>
<td>The Suspension Status is missing</td>
</tr>
<tr>
<td>-8</td>
<td>Exception when inserting member. Contains the exception generated.</td>
</tr>
<tr>
<td>-9</td>
<td>Exception when updating member. Contains the exception generated.</td>
</tr>
<tr>
<td>-10</td>
<td>Supplied Card Number belongs to a different member</td>
</tr>
<tr>
<td>-11</td>
<td>Exception when adding card. Contains the exception generated.</td>
</tr>
<tr>
<td>-12</td>
<td>Exception when updating card. Contains the exception generated.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Return String</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>-13</td>
<td>Exception when updating Member Other information step failed. Contains the exception generated.</td>
</tr>
<tr>
<td>-14</td>
<td>Exception when updating member status. Contains the exception generated.</td>
</tr>
<tr>
<td>-15</td>
<td>Error when updating member analysis. Contains the exception generated.</td>
</tr>
<tr>
<td>-16</td>
<td>Exception when attempting to update member analysis. Contains the exception generated.</td>
</tr>
<tr>
<td>-17</td>
<td>Value supplied for member status is not valid</td>
</tr>
</tbody>
</table>
5 MEMBER DELETE

5.1 Stored Procedure SP_DELETE_MEMBER

The Membership System may call the stored procedure SP_DELETE_MEMBER to delete an existing member from the S7000 system.

Request to delete a member in S7000 will delete that member, including cards, all points and history associated with that member.

Note: The Membership System has to ensure that the member details associated with a Member Number is not changed to another member's without first deleting the original Member Number. Not doing so will result in the new person inheriting any remaining promotional points and play history of the original person.

Following is the parameter structure of the stored procedure:

```sql
SP_DELETE_MEMBER
(
  p_member_number IN VARCHAR2(12), --member's number
  o_return_code OUT NUMBER,  --output return code
  o_return_str OUT VARCHAR2  --output return message
)
```

5.1.1 Calling the stored procedure

<table>
<thead>
<tr>
<th>Parameter</th>
<th>IN/OUT</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_member_number</td>
<td>IN</td>
<td>VARCHAR2(12)</td>
<td>The member number of the member who has to be deleted from the S7000 system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This parameter is mandatory.</td>
</tr>
<tr>
<td>o_return_code</td>
<td>OUT</td>
<td>NUMBER(5)</td>
<td>Output parameter from the stored procedure i.e. the return code which indicates the result of the procedure whether the member add/update process was successful or has failed as a result of incorrect data supplied or for any other reason. Refer to section 6.1.2 for detailed information.</td>
</tr>
<tr>
<td>o_return_str</td>
<td>OUT</td>
<td>VARCHAR2 (255)</td>
<td>Output parameter from the stored procedure i.e. the string that gives more information about the above return code; it indicates the result of the procedure whether the member add/update process was successful or has failed as a result of incorrect data supplied or for any other reason. Refer to section 6.1.2 for detailed information.</td>
</tr>
</tbody>
</table>

5.1.2 Checking the result of the procedure

The output code and string returned by the stored procedure sp_add_update_member will be one of the following:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Return String</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Success</td>
</tr>
<tr>
<td>0</td>
<td>Failed. Contains the exception generated by the stored procedure</td>
</tr>
</tbody>
</table>
6 GET NEXT MEMBERSHIP NUMBER

6.1 Function GET_NEXT_MEMBERNUMBER

The Membership System may call the GET_NEXT_MEMBERNUMBER to fetch the next membership number from the S7000 system.

This function may be used only in the instances where the S7000 system is required to maintain and generate membership numbers. When using this function, the Membership System therefore is required to

1. call GET_NEXT_MEMBERNUMBER to fetch the next membership number

2. call the SP_ADD_UPDATE_MEMBER_EXT or SP_ADD_UPDATE_MEMBER stored procedure using the membership number from step 1.

The function may not be used if the Membership System has a facility to maintain and supply membership numbers when calling the stored procedure SP_ADD_UPDATE_MEMBER_EXT or SP_ADD_UPDATE_MEMBER.

6.1.1 Calling the function and checking the result

The function GET_NEXT_MEMBERNUMBER returns a number which is the next membership number

If the function execution failed as a result of an exception, the return value will be -1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUT Parameter</td>
<td>NUMBER(12)</td>
<td>Returns the next member number if the function execution was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Returns -1 if the function execution failed as a result of an exception.</td>
</tr>
</tbody>
</table>
7 C# EXAMPLE

The following example code is provided as a guide to assist in development of the interface to S7000 using C#. This example might not be programatically correct.

The example has been written using The C# language in Visual Studio.NET (.Net 1.1 framework) development environment and using Oracle Data Access Components. To include this in your C# Project choose Project>References and then choose Oracle.DataAccess.dll Library as shown.

```csharp
using System;
using System.Data;
using Oracle.DataAccess.Client;
using System.Globalization;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication4
{
    class Test
    {
        /// <summary>
        /// Database Operation type
        /// </summary>
        public enum OperType
        {
            UpdateMember,  //Normal add/update operation
            UpdateMemberExt,  //Extended add/update operation
            DeleteMember  //Delete operation
        }
    }
}
```
/// Test external membership interface module
/// <summary>
/// void TestDirectly()
/// try
///  //Declare variables
/// System.Text.StringBuilder _sqlBlock = new System.Text.StringBuilder();
/// OracleConnection _conn;
/// OracleCommand _cmd;
/// OracleParameter[] _parameters;
/// DateTimeFormatInfo dtFormat = new DateTimeFormatInfo();
/// dtFormat.ShortDatePattern = "dd/MM/yyyy";
/// //Set Operation type
/// OperType operType = OperType.UpdateMemberExt;
/// //Generate the connection string
/// string _conString = String.Format("User Id={0};Password={1};Data Source={2}" , "dbusername", "dbpassword", "dbname");
/// //Create connection instance of Oracle
/// _conn = new OracleConnection(_conString);
/// //Check connection status ,if database is closed then open it
/// if (_conn != null && _conn.State == ConnectionState.Closed)
/// { _conn.Open(); }
/// //Define and set parameters value
/// if (operType == OperType.DeleteMember)
/// { //Delete Case
/// _parameters = new OracleParameter[3];
/// //Member number
/// _parameters[0] = new OracleParameter();
/// _parameters[0].ParameterName = "p_member_number";
/// _parameters[0].DbType = DbType.String;
/// _parameters[0].Size = 12;
/// _parameters[0].Direction = ParameterDirection.Input;
/// _parameters[0].Value = Convert.ToString("1234123473");
/// //Result, 0 - Success ,1 - failed
/// _parameters[1] = new OracleParameter();
/// _parameters[1].ParameterName = "o_return_code";
/// _parameters[1].DbType = DbType.Int16;
/// _parameters[1].Size = 1;
/// _parameters[1].Direction = ParameterDirection.Output;
/// //Result, message return
/// _parameters[2] = new OracleParameter();
/// _parameters[2].ParameterName = "o_return_str";
/// _parameters[2].DbType = DbType.String;
/// _parameters[2].Size = 256;
/// _parameters[2].Direction = ParameterDirection.Output;
/// }
/// else
/// { //Add/Update Case
/// _parameters = new OracleParameter[21];
/// //Member number
/// _parameters[0] = new OracleParameter();
/// _parameters[0].ParameterName = "p_member_number";
/// _parameters[0].DbType = DbType.String;
/// _parameters[0].Size = 12;
/// _parameters[0].Direction = ParameterDirection.Input;
/// _parameters[0].Value = Convert.ToString("1234123473");
/// //Member first name
/// _parameters[1] = new OracleParameter();
/// _parameters[1].ParameterName = "p_firstname";
/// _parameters[1].DbType = DbType.String;
/// _parameters[1].Size = 25;
/// _parameters[1].Direction = ParameterDirection.Input;
/// _parameters[1].Value = Convert.ToString("first name1");
/// }
//Member surname
_parameters[2] = new OracleParameter();
_parameters[2].ParameterName = "p_surname";
_parameters[2].DbType = DbType.String;
_parameters[2].Size = 25;
_parameters[2].Direction = ParameterDirection.Input;
_parameters[2].Value = Convert.ToString("surname");

//Member title
_parameters[3] = new OracleParameter();
_parameters[3].ParameterName = "p_title";
_parameters[3].DbType = DbType.String;
_parameters[3].Size = 25;
_parameters[3].Direction = ParameterDirection.Input;
_parameters[3].Value = Convert.ToString("Mr.");

//Member full day of birthday
_parameters[4] = new OracleParameter();
_parameters[4].ParameterName = "p_date_of_birth";
_parameters[4].DbType = DbType.DateTime;
_parameters[4].Direction = ParameterDirection.Input;
_parameters[4].Value = DateTime.Parse("18/08/1980", dtFormat);

//Member joined/registered date
_parameters[5] = new OracleParameter();
_parameters[5].ParameterName = "p_join_date";
_parameters[5].DbType = DbType.DateTime;
_parameters[5].Direction = ParameterDirection.Input;
_parameters[5].Value = DateTime.Parse("28/04/2009", dtFormat);

//Gender
_parameters[6] = new OracleParameter();
_parameters[6].ParameterName = "p_gender";
_parameters[6].DbType = DbType.String;
_parameters[6].Size = 1;
_parameters[6].Direction = ParameterDirection.Input;
_parameters[6].Value = Convert.ToString("M");

//Telephone
_parameters[7] = new OracleParameter();
_parameters[7].ParameterName = "p_telephone";
_parameters[7].DbType = DbType.String;
_parameters[7].Size = 20;
_parameters[7].Direction = ParameterDirection.Input;
_parameters[7].Value = Convert.ToString("0290129133");

//Mobile number
_parameters[8] = new OracleParameter();
_parameters[8].ParameterName = "p_mobile_phone";
_parameters[8].DbType = DbType.String;
_parameters[8].Size = 20;
_parameters[8].Direction = ParameterDirection.Input;
_parameters[8].Value = Convert.ToString("0432433433");

//Member Address 1
_parameters[9] = new OracleParameter();
_parameters[9].ParameterName = "p_address_line1";
_parameters[9].DbType = DbType.String;
_parameters[9].Size = 60;
_parameters[9].Direction = ParameterDirection.Input;
_parameters[9].Value = Convert.ToString("Address Line 1");

//Member Address 2
_parameters[10] = new OracleParameter();
_parameters[10].ParameterName = "p_address_line2";
_parameters[10].DbType = DbType.String;
_parameters[10].Size = 60;
_parameters[10].Direction = ParameterDirection.Input;
_parameters[10].Value = Convert.ToString("Address Line 2");

//Member Address 3
_parameters[11].ParameterName = "p_address_line3";
_parameters[11].DbType = DbType.String;
_parameters[11].Size = 60;
```csharp
//Member Address 4
_parameters[12] = new OracleParameter();
_parameters[12].ParameterName = "p_address_line4";
_parameters[12].DbType = DbType.String;
_parameters[12].Size = 60;
_parameters[12].Direction = ParameterDirection.Input;
_parameters[12].Value = Convert.ToString("Address Line 4");

//Member Address post code
_parameters[13] = new OracleParameter();
_parameters[13].ParameterName = "p_post_code";
_parameters[13].DbType = DbType.String;
_parameters[13].Size = 12;
_parameters[13].Direction = ParameterDirection.Input;
_parameters[13].Value = Convert.ToString("1123");

//Barcode
_parameters[14] = new OracleParameter();
_parameters[14].ParameterName = "p_mail_barcode";
_parameters[14].DbType = DbType.String;
_parameters[14].Size = 37;
_parameters[14].Direction = ParameterDirection.Input;
_parameters[14].Value = Convert.ToString("1234");

//Email address
_parameters[15] = new OracleParameter();
_parameters[15].ParameterName = "p_email";
_parameters[15].DbType = DbType.String;
_parameters[15].Size = 64;
_parameters[15].Direction = ParameterDirection.Input;
_parameters[15].Value = Convert.ToString("test@testemail.ali.com.au");

//Member card number
_parameters[16] = new OracleParameter();
_parameters[16].ParameterName = "p_card_number";
_parameters[16].DbType = DbType.String;
_parameters[16].Size = 22;
_parameters[16].Direction = ParameterDirection.Input;
_parameters[16].Value = Convert.ToString("1234123473");

//Card expired date
_parameters[17] = new OracleParameter();
_parameters[17].ParameterName = "p_card_expiry_date";
_parameters[17].DbType = DbType.DateTime;
_parameters[17].Direction = ParameterDirection.Input;
_parameters[17].Value = DateTime.Parse("28/04/2011", dtFormat);

//Suspension flag
_parameters[18] = new OracleParameter();
_parameters[18].ParameterName = "p_suspension_status";
_parameters[18].DbType = DbType.Int16;
_parameters[18].Size = 1;
_parameters[18].Direction = ParameterDirection.Input;
_parameters[18].Value = Convert.ToInt16("0");

//Result, 0 - Success, 1 - failed
_parameters[19] = new OracleParameter();
_parameters[19].ParameterName = "o_return_code";
_parameters[19].DbType = DbType.Int16;
_parameters[19].Size = 1;
_parameters[19].Direction = ParameterDirection.Output;

//Result, message return
_parameters[20] = new OracleParameter();
_parameters[20].ParameterName = "o_return_str";
_parameters[20].DbType = DbType.String;
_parameters[20].Size = 256;
_parameters[20].Direction = ParameterDirection.Output;

//member status
_parameters[21] = new OracleParameter();
_parameters[21].ParameterName = "p_mem_status";
_parameters[21].DbType = DbType.Int16;
_parameters[21].Size = 1;
_parameters[21].Direction = ParameterDirection.Input;
_parameters[21].Value = Convert.ToInt16("0");

} //Generate sql for extended member analysis operation
```
if (operType == OperType.UpdateMemberExt)
{
    System.Text.StringBuilder _analysisSql;
    String[,] analysis = new String[100, 2];
    //Member analysis type and Value
    _analysisSql = new System.Text.StringBuilder();
    //Analysis Type code of the Nationality group
    analysis[0, 0] = "02";
    //Analysis Value code for the Nationality of the member
    analysis[0, 1] = Convert.ToString(5);
    //Analysis Type code of the Nationality group
    analysis[1, 0] = "02";
    //Analysis Value code for the Nationality of the member
    analysis[1, 1] = Convert.ToString(10);
    //Analysis Type code for the Membership card type group
    analysis[2, 0] = "05";
    //Analysis Value code for the card type held by the member
    analysis[2, 1] = Convert.ToString(25);

    _analysisSql.Append(String.Format("P_ANALYSIS_ARRAY(1)(1) := {0};
P_ANALYSIS_ARRAY(1)(2) := {1};
P_ANALYSIS_ARRAY(2)(1) := {2};
P_ANALYSIS_ARRAY(2)(2) := {3};
P_ANALYSIS_ARRAY(3)(1) := {4};
P_ANALYSIS_ARRAY(3)(2) := {5};", 
        analysis[0, 0],
        analysis[0, 1],
        analysis[1, 0],
        analysis[1, 1],
        analysis[2, 0],
        analysis[2, 1]));

    //Generate the sql text
    _sqlBlock.Append("DECLARE P_ANALYSIS_ARRAY SGM.twodim_aa.array_t;BEGIN ");
    _sqlBlock.Append(_analysisSql.ToString());
    _sqlBlock.Append("SGM.SP_ADD_UPDATE_MEMBER_EXT(:p_member_number,
        :p_telephone, :p_mobile_phone, :p_address_line1, :p_address_line2, :p_address_line3,
        :p_address_line4, :p_post_code, :p_mail_barcode, :p_email, :p_card_number,
        :p_card_expiry_date, :p_suspension_status, P_ANALYSIS_ARRAY, :o_return_code,
        :o_return_str,:p_mem_status);COMMIT;END;"};
}

//Creat instance for OracleCommand
    _cmd = new OracleCommand();
    //Set to current connect
    _cmd.Connection = _conn;
    //Command type
    //Stored procedure name
    if (operType == OperType.DeleteMember)
    {
        _cmd.CommandType = CommandType.StoredProcedure;
        _cmd.CommandText = "SP_DELETE_MEMBER";
    }

    else if (operType == OperType.UpdateMember)
    {
        _cmd.CommandType = CommandType.StoredProcedure;
        _cmd.CommandText = "SP_ADD_UPD_TE_MEMBER";
    }

    else if (operType == OperType.UpdateMemberExt)
    {
        _cmd.CommandType = CommandType.Text;
        //Execute the stored procedure SP_ADD_UPDATE_MEMBER_EXT using sql text
        _cmd.CommandText = _sqlBlock.ToString();
    }

    //Initialize parameters of OracleCommand
    if (_cmd.Parameters.Count > 0)
    {
        _cmd.Parameters.Clear();
    }
```csharp
foreach (OracleParameter param in _parameters)
{
    // Check null for the parameter value
    if ((param.Direction == ParameterDirection.InputOutput) && (param.Value == null))
    {
        param.Value = DBNull.Value;
    }
    // Add parameter in OracleCommand
    _cmd.Parameters.Add(param);
}

// Execute SQL query and return result
int result = _cmd.ExecuteNonQuery();

if (operType == OperType.DeleteMember)
else

// Clear parameters of OracleCommand
_cmd.Parameters.Clear();

// Release allocated resources
foreach (OracleParameter current in _parameters)
{
    current.Dispose();
}

// Check connection status, if database is opened then close it
if (_conn != null && _conn.State == ConnectionState.Open)
{
    _cmd.Dispose();
    _conn.Close();
    _conn.Dispose();
}
}

} catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```
8 VB.NET Example

The following example code is provided as a guide to assist in development of the interface to S7000 using VB.Net. This example might not be programmatically correct.

The example has been written using The VB.Net language in Visual Studio.NET (.Net 1.1 Framework) development environment and using Oracle Data Access Components. To include this in your VB.Net Project choose Project>References and then choose Oracle.DataAccess.dll Library as shown.

Imports Oracle.DataAccess.Client
Imports System
Imports System.Data
Imports System.Globalization
Public Class Test
  '''<summary>
  ''' Database Operation type
  '''</summary>
  Public Enum OperType
    UpdateMember 'Normal add/update operation
    UpdateMemberExt 'Extended add/update operation
    DeleteMember 'Delete operation
  End Enum

  ''' <summary>
  ''' Test external membership interface module
  '''</summary>
  Public Sub TestDirectly()
    Try
      Declare variables
      Dim _conn As OracleConnection

      ' Further code...
    Catch
      ' Handle exceptions
    End Try
  End Sub
End Class
Dim _cmd As OracleCommand
Dim _parameters As OracleParameter()
Dim operType As OpertType
Dim dtFormat As New DateTimeFormatInfo
Dim index As Integer
Dim _sqlBlock As New System.Text.StringBuilder

dtFormat.ShortDatePattern = "dd/MM/yyyy"

'Set Operation type
operType = operType.DeleteMember

'Generate the connection string
Dim _conString As String = String.Format("User Id={0};Password={1};Data Source={2}", "dbusername", "dbpassword", "dbname")

'Create connection instance of Oracle
_conn = New OracleConnection(_conString)

'Check connection status, if database is closed then open it
If ((_conn Is Nothing) Or (_conn.State = ConnectionState.Closed)) Then
    _conn.Open()
End If

'Define and set parameters value
If (operType = operType.DeleteMember) Then 'Delete Case
    ReDim _parameters(2)
    'Member number
    _parameters(0) = New OracleParameter
        _parameters(0).ParameterName = "p_member_number"
        _parameters(0).DbType = DbType.String
        _parameters(0).Size = 12
        _parameters(0).Direction = ParameterDirection.Input
        _parameters(0).Value = Convert.ToString("1234123471")
    'Result, 0 - Success ,1 - failed
    _parameters(1) = New OracleParameter
        _parameters(1).ParameterName = "o_return_code"
        _parameters(1).DbType = DbType.Int16
        _parameters(1).Size = 1
        _parameters(1).Direction = ParameterDirection.Output
    'Result, message
    _parameters(2) = New OracleParameter
        _parameters(2).ParameterName = "o_return_str"
        _parameters(2).DbType = DbType.String
        _parameters(2).Size = 256
        _parameters(2).Direction = ParameterDirection.Output
Else 'Add/Update Case
    ReDim _parameters(20)
    'Member number
    _parameters(0) = New OracleParameter
        _parameters(0).ParameterName = "p_member_number"
        _parameters(0).DbType = DbType.String
        _parameters(0).Size = 12
        _parameters(0).Direction = ParameterDirection.Input
        _parameters(0).Value = Convert.ToString("1234123471")
    'Member first name
    _parameters(1) = New OracleParameter
        _parameters(1).ParameterName = "p_firstname"
        _parameters(1).DbType = DbType.String
        _parameters(1).Size = 25
        _parameters(1).Direction = ParameterDirection.Input
        _parameters(1).Value = Convert.ToString("first name")
    'Member surname
    _parameters(2) = New OracleParameter
        _parameters(2).ParameterName = "p_surname"
        _parameters(2).DbType = DbType.String
        _parameters(2).Size = 25
        _parameters(2).Direction = ParameterDirection.Input
        _parameters(2).Value = Convert.ToString("surname")
    'Member title
    _parameters(3) = New OracleParameter
_parameters(3).ParameterName = "p_title"
_parameters(3).DbType = DbType.String
_parameters(3).Size = 25
_parameters(3).Direction = ParameterDirection.Input
_parameters(3).Value = Convert.ToString("Mr.")

'Member full day of birthday
_parameters(4) = New OracleParameter
_parameters(4).ParameterName = "p_date_of_birth"
_parameters(4).DbType = DbType.DateTime
_parameters(4).Direction = ParameterDirection.Input

'Member joined/registered date
_parameters(5) = New OracleParameter
_parameters(5).ParameterName = "p_join_date"
_parameters(5).DbType = DbType.DateTime
_parameters(5).Direction = ParameterDirection.Input
_parameters(5).Value = DateTime.Parse("28/04/2009", dtFormat)

'Gender
_parameters(6) = New OracleParameter
_parameters(6).ParameterName = "p_gender"
_parameters(6).DbType = DbType.String
_parameters(6).Size = 1
_parameters(6).Direction = ParameterDirection.Input
_parameters(6).Value = Convert.ToString("M")

'Telephone
_parameters(7) = New OracleParameter
_parameters(7).ParameterName = "p_telephone"
_parameters(7).DbType = DbType.String
_parameters(7).Size = 20
_parameters(7).Direction = ParameterDirection.Input
_parameters(7).Value = Convert.ToString("0290129133")

'Mobile number
_parameters(8) = New OracleParameter
_parameters(8).ParameterName = "p_mobile_phone"
_parameters(8).DbType = DbType.String
_parameters(8).Size = 20
_parameters(8).Direction = ParameterDirection.Input
_parameters(8).Value = Convert.ToString("0432433433")

'Member Address 1
_parameters(9) = New OracleParameter
_parameters(9).ParameterName = "p_address_line1"
_parameters(9).DbType = DbType.String
_parameters(9).Size = 60
_parameters(9).Direction = ParameterDirection.Input
_parameters(9).Value = Convert.ToString("Address Line 1")

'Member Address 2
_parameters(10) = New OracleParameter
_parameters(10).ParameterName = "p_address_line2"
_parameters(10).DbType = DbType.String
_parameters(10).Size = 60
_parameters(10).Direction = ParameterDirection.Input
_parameters(10).Value = Convert.ToString("Address Line 2")

'Member Address 3
_parameters(11) = New OracleParameter
_parameters(11).ParameterName = "p_address_line3"
_parameters(11).DbType = DbType.String
_parameters(11).Size = 60
_parameters(11).Direction = ParameterDirection.Input
_parameters(11).Value = Convert.ToString("Address Line 3")

'Member Address 4
_parameters(12) = New OracleParameter
_parameters(12).ParameterName = "p_address_line4"
_parameters(12).DbType = DbType.String
_parameters(12).Size = 60
_parameters(12).Direction = ParameterDirection.Input
_parameters(12).Value = Convert.ToString("Address Line 4")

'Member Address post code
_parameters(13) = New OracleParameter
_parameters(13).ParameterName = "p_post_code"
_parameters(13).DbType = DbType.String
End If

' Generate sql for extended member analysis operation
If (operType = operType.UpdateMemberExt) Then
    Dim _analysisSql As New System.Text.StringBuilder
    Dim analysis(100, 2) As String
    'Member analysis type and Value
    'Analysis Type code of the Nationality group
    analysis(0, 0) = "02"
    'Analysis Value code for the Nationality of the member
    analysis(0, 1) = Convert.ToString(5)
analysis(1, 0) = "02"

'Analysis Value code for the Nationality of the member
analysis(1, 1) = Convert.ToString(10)

'Analysis Type code for the Membership card type group
analysis(2, 0) = "05"

'Analysis Value code for the card type held by the member
analysis(2, 1) = Convert.ToString(25)

'Generate array in sql for member analysis
_analysisSql.Append(String.Format("P_ANALYSIS_ARRAY(1)(1) := {0};", analysis(0, 0)))
_analysisSql.Append(String.Format("P_ANALYSIS_ARRAY(1)(2) := {0};", analysis(0, 1)))
_analysisSql.Append(String.Format("P_ANALYSIS_ARRAY(2)(1) := {0};", analysis(1, 0)))
_analysisSql.Append(String.Format("P_ANALYSIS_ARRAY(2)(2) := {0};", analysis(1, 1)))
_analysisSql.Append(String.Format("P_ANALYSIS_ARRAY(3)(1) := {0};", analysis(2, 0)))
_analysisSql.Append(String.Format("P_ANALYSIS_ARRAY(3)(2) := {0};", analysis(2, 1)))

'Generate the sql text
_sqlBlock.Append("DECLARE P_ANALYSIS_ARRAY SGM.twodim_aa.array_t;BEGIN ")
_sqlBlock.Append(_analysisSql.ToString())
_sqlBlock.Append("SGM.SP_ADD_UPDATE_MEMBER_EXT(:p_member_number, ":p_firstname, ":p_surname, ":p_title, ":p_date_of_birth, ":p_join_date, ":p_gender, ":p_telephone, ":p_mobile_phone, ":p_address_line1, ":p_address_line2, ":p_address_line3, ":p_address_line4, ":p_post_code, ":p_mail_barcode, ":p_email, ":p_card_number, ":p_card_expiry_date, ":p_suspension_status, P_ANALYSIS_ARRAY, :o_return_code, ":o_return_str, :p_mem_status);COMMIT;END;"
End If

'Create instance for OracleCommand
_cmd = New OracleCommand
'Set to current connect
_cmd.Connection = _conn

'Stored procedure name
If (operType = operType.DeleteMember) Then
'Command type
_cmd.CommandType = CommandType.StoredProcedure
_cmd.CommandText = "SP_DELETE_MEMBER"
ElseIf (operType = operType.UpdateMember) Then
'Command type
_cmd.CommandType = CommandType.StoredProcedure
_cmd.CommandText = "SP_ADD_UPDATE_MEMBER"
ElseIf (operType = operType.UpdateMemberExt) Then
'Command type
_cmd.CommandType = CommandType.Text
'Throw the stored procedure SP_ADD_UPDATE_MEMBER_EXT using sql text
_cmd.CommandText = _sqlBlock.ToString()
End If

'Initialize parameters of OracleCommand
If (_cmd.Parameters.Count > 0) Then
_cmd.Parameters.Clear()
End If

'Check and add parameters in OracleCommand
If ((Not _parameters Is Nothing) And (_parameters.Length > 0)) Then
    For Each param As OracleParameter In _parameters
        'Check null for the parameter value
        If (param.Direction = ParameterDirection.InputOutput) And
            (param.Value = Nothing) Then
            param.Value = DBNull.Value
        End If
    Next param
End If

'Execute sql query and return result
Dim result As Integer = _cmd.ExecuteNonQuery()

If (operType = operType.DeleteMember) Then
    Console.WriteLine(String.Format("Delete {0}:{1}", Convert.ToInt16(_parameters(1).Value), Convert.ToString(_parameters(2).Value)))
Else
End If

'Clear parameters of OracleCommand
_cmd.Parameters.Clear()

'Release allocated resources
For Each current As OracleParameter In _parameters
    current.Dispose()
Next

'Check connection status, if database is opened then close it
If (_conn Is Nothing) And (_conn.State = ConnectionState.Open) Then
    _cmd.Dispose()
    _conn.Close()
    _conn.Dispose()
End If

Catch ex As Exception
    Console.WriteLine(ex.Message)
End Try
End Sub

Public Sub New()

End Sub
End Class
9 Visual Basic Example

The following example code is provided as a guide to assist in development of the interface to S7000 using Visual Basic 6.0. This example might not be programmatically correct.

The example has been written using The Visual Basic 6.0 development environment and using ActiveX Data Objects Version 2.6. To include this in your Visual Basic Project choose Project>References and then choose Microsoft ActiveX Data Objects 2.6 Library as shown.

```
Public Enum OperType 'DB Operation Type
    UpdateMember
    UpdateMemberExt
    DeleteMember
End Enum

Public Sub TestDirectlyForADO()
    'Declare variables
    Dim conn As New ADODB.Connection
    Dim cmd As New ADODB.Command
    Dim params As ADODB.Parameters
    Dim param As ADODB.Parameter
    Dim opType As OperType
    'Specify connection string
    Dim mConnectionString As String
    Dim sqlBlock As String
    Dim analysis(IOO, 2) As String
    Dim analysisSql As String
    Dim index As Integer

    opType = OperType.UpdateMemberExt

    mConnectionString = "DSN=" & Trim("dbname") & ";" & "UID=" & Trim("dbusername") & ";PWD=" & Trim("dbpassword")
    conn.ConnectionString = mConnectionString
    'Check connection status and try to open the database
    If conn.State = adStateClosed Then
        conn.Open
    End If

    'Create and define command
    Set cmd = New ADODB.Command
    cmd.ActiveConnection = conn
    'set active Connection
```
If (opType = OperType.DeleteMember) Then
    cmd.CommandText = "SP_DELETE_MEMBER" 'Set command text to stored procedure name
    cmd.CommandType = adCmdStoredProc 'Set command type to stored procedure
ElseIf (opType = OperType.UpdateMember) Then
    cmd.CommandText = "SP_ADD_UPDATE_MEMBER" 'Set command text to stored procedure name
    cmd.CommandType = adCmdStoredProc 'Set command type to stored procedure
ElseIf (opType = OperType.UpdateMemberExt) Then
    cmd.CommandType = adCmdText 'Anonymous block
    cmd.Prepared = True
End If

Set params = cmd.Parameters 'Create and define command parameters

If (opType = OperType.DeleteMember) Then
    'Define stored procedure parameters and append to command
    params.Append cmd.CreateParameter("p_member_number", adVarChar, adParamInput, 12)
    params.Append cmd.CreateParameter("o_return_code", adInteger, adParamOutput)
    params.Append cmd.CreateParameter("o_return_str", adVarChar, adParamOutput, 256)
Else
    'Define stored procedure parameters and append to command
    params.Append cmd.CreateParameter("p_member_number", adVarChar, adParamInput, 12)
    params.Append cmd.CreateParameter("p_surname", adVarChar, adParamInput, 25)
    params.Append cmd.CreateParameter("p_date_of_birth", adDate, adParamInput)
    params.Append cmd.CreateParameter("p_join_date", adDate, adParamInput)
    params.Append cmd.CreateParameter("p_gender", adVarChar, adParamInput, 1)
    params.Append cmd.CreateParameter("p_phone", adVarChar, adParamInput, 20)
    params.Append cmd.CreateParameter("p_mobile_phone", adVarChar, adParamInput, 20)
    params.Append cmd.CreateParameter("p_address_line1", adVarChar, adParamInput, 60)
    params.Append cmd.CreateParameter("p_address_line2", adVarChar, adParamInput, 60)
    params.Append cmd.CreateParameter("p_address_line3", adVarChar, adParamInput, 60)
    params.Append cmd.CreateParameter("p_address_line4", adVarChar, adParamInput, 60)
    params.Append cmd.CreateParameter("p_post_code", adVarChar, adParamInput, 12)
    params.Append cmd.CreateParameter("p_mail_barcode", adVarChar, adParamInput, 37)
    params.Append cmd.CreateParameter("p_email", adVarChar, adParamInput, 64)
    params.Append cmd.CreateParameter("p_card_number", adVarChar, adParamInput, 22)
    params.Append cmd.CreateParameter("p_card_expiry_date", adDate, adParamInput)
    params.Append cmd.CreateParameter("p_suspension_status", adInteger, adParamInput)
    params.Append cmd.CreateParameter("p_mem_status", adInteger, adParamInput)
Else
    'Specify input parameters
    params("p_member_number").Value = "1234123468"
    params("p_firstname").Value = "first name2"
    params("p_surname").Value = "surname"
    params("p_title").Value = "Mr."
    params("p_date_of_birth").Value = DateSerial(1980, 8, 18)
    params("p_join_date").Value = DateSerial(2009, 4, 28)
    params("p_gender").Value = "M"
    params("p_phone").Value = "0290229022"
    params("p_mobile_phone").Value = "0432432432"
    params("p_address_line1").Value = "Address Line 1"
    params("p_address_line2").Value = "Address Line 2"
    params("p_address_line3").Value = "Address Line 3"
    params("p_address_line4").Value = "Address Line 4"
    params("p_post_code").Value = "2223"
    params("p_mail_barcode").Value = "123432424"
    params("p_email").Value = "email@test.ali.com.au"
    params("p_card_number").Value = "1234123468"
    params("p_card_expiry_date").Value = DateSerial(2011, 8, 18)
    params("p_suspension_status").Value = 0
    params("p_mem_status").Value = 0
Else
    'Specify input parameters
    params("p_member_number").Value = "1234123468"
    params("p_firstname").Value = "first name2"
    params("p_surname").Value = "surname"
    params("p_title").Value = "Mr."
    params("p_date_of_birth").Value = DateSerial(1980, 8, 18)
    params("p_join_date").Value = DateSerial(2009, 4, 28)
Else
    'Member analysis type and Value
    analysis(0, 0) = "02" 'Analysis Type code of the Nationality group
    analysis(0, 1) = Str(5) 'Analysis Value code for the Nationality of the member
End If
analysis(1, 0) = "02" 'Analysis Type code of the Nationality group
analysis(1, 1) = Str(10) 'Analysis Value code for the Nationality of the member
analysis(2, 0) = "05" 'Analysis Type code for the Membership card type group
analysis(2, 1) = Str(25) 'Analysis Value code for the card type held by the member

'Generate array in sql for member analysis
analysisSql = "P_ANALYSIS_ARRAY(1)(1) := " & analysis(0, 0) & ";" & _
"P_ANALYSIS_ARRAY(1)(2) := " & analysis(0, 1) & ";" & _
"P_ANALYSIS_ARRAY(2)(1) := " & analysis(1, 0) & ";" & _
"P_ANALYSIS_ARRAY(2)(2) := " & analysis(1, 1) & ";" & _
"P_ANALYSIS_ARRAY(3)(1) := " & analysis(2, 0) & ";" & _
"P_ANALYSIS_ARRAY(3)(2) := " & analysis(2, 1) & ";"

'Populate sql command text
sqlBlock = "DECLARE P_ANALYSIS_ARRAY SGM.twodim_aa.array_t;BEGIN " & _

cmd.CommandText = sqlBlock
End If

'Execute the command
cmd.Execute

'Display return value
MsgBox (params("o_return_code").Value & ":" & params("o_return_str").Value)

'Check connection status and close the DB connection
If conn.State = adStateOpen Then
conn.Close
End If

End Sub
## 10 DOCUMENT CHANGE HISTORY

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Changes made</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>06-05-09</td>
<td>Initial Release</td>
</tr>
<tr>
<td>0.2</td>
<td>13-08-10</td>
<td>Included section 4.2 for Member Analysis Enhancement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated all the code examples to include the extension SP call</td>
</tr>
<tr>
<td>0.3</td>
<td>10-09-10</td>
<td>Changes for the additional parameter member status</td>
</tr>
<tr>
<td>0.4</td>
<td>21-06-12</td>
<td>Added GET_NEXT_MEMBERNUMBER</td>
</tr>
</tbody>
</table>

[END OF DOCUMENT]