API Instructions

The WSDL files for the main API can be found by navigating to:

https://portal.agmonitoring.com/teststagesexternalgateway/gateway.asmx (sandbox)
https://portal.agmonitoring.com/stagesexternalgateway/gateway.asmx (production)

The most efficient way to use the API is by referencing the following DLL's in the project. They have already serialized the web service and significantly increase startup time.

HTTPServiceLibrary.dll
StagesGateway.dll

To utilize the precompiled serializers, make the following call:

```
Private stagesapi As Stages.StagesGateway
stagesapi = New Stages.StagesGateway(My.Settings.StagesPortal)
```

The StagesPortal setting will need to be application configurable so we can switch between sandbox and production. Additionally, the referenced address needs to be the above API addresses changed to '.aspx' instead of '.asmx.'

Login and Logout

SGS's Stages handles the authentication of users. This is accomplished through the 'Login' API call:

```
Private si As Stages.SessionInfo = Nothing
Dim lqr As Stages.QueryResult(Of Stages.EmptyResult, Stages.SessionInfo) =
stagesapi.Login(userName:=My.Settings.Username, password:=My.Settings.Password)
si = lqr.OutputParameter
```

The SessionInfo class contains the authenticated user's 'SessionNum' and 'SessionPassword' used for all other calls. When logging out, simply utilize the SessionNum and SessionPassword to end the session:

```
Dim lqr As Stages.QueryResult(Of Stages.EmptyResult, Stages.EmptyResult) =
stagesapi.Logout(si.SessionNum, si.SessionPassword, False, "")
 si = Nothing
```

Validating API Success

For whatever reason, 'Success' in the API result does not necessarily mean everything worked great (nor does ErrorTypeNum=0 do so. The following code is what I use to determine whether an error occurred as part of an API call:

```
Friend Shared Sub CheckQueryResult(ByVal rslt As Stages.BaseQueryResult, ByVal queryaction As
String)
If Not rslt.Success And Not rslt.ErrorTypeNum = 124 Then
        Throw New ApplicationException(String.Format("Error {0}; ErrorTypeNum: {1}; Message:
        {2}", queryaction, rslt.ErrorTypeNum, rslt.UserErrorMessage))
End If
```

```
End Sub
```

If I were to use the CheckQueryResult for the login code, after making the call to the API call I would use the following line:

```
CheckQueryResult(lqr, "login")
```

Search

To search for an account, simply pass either the SiteName or the TransmitterCode to the XtAdvancedSearch API method:

```
Dim xt As Stages.XtAdvancedSearchQueryResult =
stagesapi.XtAdvancedSearch(sessionNum:=si.SessionNum, sessionPassword:=si.SessionPassword,
TransmitterCode:=txtText.Text)
```

The ResultSet parameter of the XtAdvancedSearchQueryResult will contain a list of the accounts that match the query. If one account matches, go straight to the account screen, otherwise show the results.

Selecting, Inserting and Updating Account Information

The accompanying excel spreadsheet displays the API calls necessary to select, insert and update the displayed fields. I will provide a simple example of each API referenced in the document (except for the delete methods as those are fairly self-explanatory).

SiteDetail

```
Dim sd As Stages.SiteDetailQueryResult = stagesapi.SiteDetail(si.SessionNum,
si.SessionPassword, [SiteNum])
```

SiteInsert

I have never made this particular call—but it appears to be a mix of two calls. Working with the vendor to clarify the procedure on this call.

SiteUpdate

```
Dim su As Stages.SiteUpdateQueryResult = stagesapi.SiteUpdate(si.SessionNum,
si.SessionPassword, [SiteNum], "AccountPortal", ColumnSiteName:="Dyer, Rob",
ChangeFlagSiteName:=True)
```

The SiteUpdate has a large number of available fields. You can specify whether to update a particular field or not as part of the call (a corresponding flag field exists for all updateable parameters). I assume that if the field isn't specified it is not updated either. To empty a field, update it with a null value. Only the fields in the UI need to be sent along with the following additions:

ColumnRegion: the US State from the address (2 digit)
SiteType: M (for this particular package)

Updating a device/Changing a TransmitterCode

ValidateXmitEntry

When adding a new account (and specifying the account number to be used) or changing a TransmitterCode (account number) you need to make sure the requested account number is available. Instead of waiting to see if the method fails, validate the new TransmitterCode first using the ValidateXmitEntry. (This also helps avoid certain fat-fingered transmitter code mistakes.)

```
Dim vxe As Stages.ValidateXmitEntryQueryResult =
stagesapi.ValidateXmitEntry(si.SessionNum, si.SessionPassword, Me.txtText.Text)
```

Three outcomes are possible from this call:

1. The transmitter code is available for use
    a. Success=True; OutputParamter.XmitOk='Ok'; ErrorTypeNum=0; UserErrorMessage=Nothing
2. The transmitter code is assigned for someone else:
    a. Success=False;OutputParamter.XmitOk=Nothing; ErrorTypeNum=0; UserErrorMessage='This Xmit# is assigned to another Site Group.'
3. The transmitter code is already in use:
    a. Success=False; OutputParameter.XmitOk=Nothing; ErrorTypeNum=0; UserErrorMessage='Duplicate Not Allowed'

DeviceCopy

Device copy allows the creation of a new account (or multiple new accounts) based on the settings of a template account. This is the recommended method of creating accounts. Make sure AvantGuard has set aside an account block for you and provided you a 'template' account. Use that transmitter code to create new accounts in the range provided. To create one account the 'StartTransmitterCode' and 'EndTransmitterCode' number will be the same. To create more, provide an 'EndTransmitterCode' value that is incrementally higher than the 'StartTransmitterCode'. Exercise care when choosing transmitter codes—one already used number will result in the entire call failing. We recommend you create one account at a time as needed.

```
Dim dc As Stages.DeviceCopyQueryResult = stagesapi.DeviceCopy(si.SessionNum,
si.SessionPassword, [TemplateAccountTransmitterCode], [StartTransmitterCode],
[EndTransmitterCode], CopySiteGroupsFlag:=True, CopyMailAddressesFlag:=True,
CopyDeviceConfigFlag:=True, CopySiteDispatchTypesFlag:=True)
```

DeviceUpdate

The device update method allows updating a variety of fields available on the device. This call can also update the TransmitterCode (commonly known as the account number). The list of parameters available is long so I refer you to the WSDL for additional information. The call below updates the transmitter code of a device.

```
Dim du As Stages.DeviceUpdateQueryResult = stagesapi.DeviceUpdate(si.SessionNum,
si.SessionPassword, [DevNum], [SiteNum], ColumnTransmitterCode:=[NewTransmitterCode],
ChangeFlagTransmitterCode:=True)
```

Activating/Cancelling a Device (Account)

To activate or cancel an account, all devices on the account must be placed 'Out of Service' (OOS). Do do this, you need to use the DeviceInOutService API method:

```
Dim io As Stages.DeviceInOutServiceQueryResult =
stagesapi.DeviceInOutService(si.SessionNum, si.SessionPassword, [DevNum], [SiteNum],
[OOSCat], Now, ConfirmXmit:=[TransmitterCode])
```

The values for [OOSCat] are as follows:

1. NULL = make the site active
2. CAN = cancel the device
3. PI = place the device as pending installation; (for brand new accounts accounts to be activated in the near future)

Selecting, Inserting, Updating and Deleting Phone Numbers
Use for both Accounts and contacts

PhoneList

```
Dim pl As Stages.PhoneListQueryResult = stagesapi.PhoneList(si.SessionNum,
si.SessionPassword, [KeyNum], [KeyType])
```

The phone number returned is formatted like (###)###-####. To request account phone numbers, send 'Site' as the 'KeyType' with the appropriate 'SiteNum' as 'KeyNum'. For a contact's phone number(s) send 'Contact' as the 'KeyType' with the 'ContactNum' as the 'KeyNum'. The possible phone types we are using in the UI will be 'C' for Cell, 'H' for home, 'W' for work. An account phone number will have a 'P' phone type for premise.

PhoneDetail

Returns all pertinent information of a particular phone number based on the:

KeyNum: [ContactNum or SiteNum]
KeyType: ['Contact' or 'Site']
SeqNum: [sequence number of the phone number]

PhoneInsert

```
Dim pi As Stages.PhoneInsertQueryResult=stagesapi.PhoneInsert(si.SessionNum,
si.SessionPassword, [KeyNum], [KeyType], [SeqNum], [Phone],[PhoneType], [AutoNotifyFlag])
```

KeyNum: primary key SiteNum for account; primary key ContactNum for contacts
KeyType: 'Site' for account; 'Contact' for contacts
PhoneNumber: ten digit phone number with no formatting
PhoneType: [For contacts: 'C' for cell; 'H' for home; 'W' for work; For Account: 'P' (for premise)]
AutoNotifyFlag: [If phone type is 'C' and user wants text notifications, set to 'True', else set to false]

PhoneUpdate

```
Dim pu As Stages.PhoneUpdateQueryResult=stagesapi.PhoneUpdate(si.SessionNum,
si.SessionPassword, pkKeyNum:=,pkKeyType:=, pkSeqNum:=,...)
```

Like the other update calls, each column has an associated 'Column Update Flag' field. Send the phone number as a ten digit number with no formatting.

PhoneDelete
Deletes a phone number given the primary key fields KeyNum, KeyType, SeqNum. Can use PhoneDeleteList to pass a collection of primary key fields to delete.

PhoneOrder
It takes the primary key of two rows in phone, reverses their order, and returns the updated seqnums

User Defined Fields

UDFValueDetail
Returns the value of the specified item from the account UDF fields. The call is fairly self explanatory. The following fields provide a guide:

KeyNum: [the SiteNum of the account]
KeyType: 'Site'
UDFCode: [one of the following values listed below depending on the value requested]

Field	UDFCode
Township/Borough	TB
Lockbox Combination	Lockbox
Hidden Key	HiddenKey
Special Notes	SpecialInstruction

UDFValueUpdate
Inserts, Updates and Deletes UDF values. Simply pass the required parameters with the UDFValue field being the desired value to store. (See above table for UDFCodes)

Status, Equipment and Other Device Information
DeviceDetail

```
Dim dd As Stages.DeviceDetailQueryResult = stagesapi.DeviceDetail(si.SessionNum,
si.SessionPassword, [DevNum], [SiteNum])
```

Contact Select, Insert, Update and Delete

ContactList

```
Dim cl As Stages.ContactListQueryResult = stagesapi.ContactList(si.SessionNum,
si.SessionPassword, [SiteNum], "Site")
```

For Package A, contacts without a SeqNum are the Subscribers. All other contacts should be listed in the contacts section in order of SeqNum.

ContactInsert
Used for both inserting a contact as well as a subscriber. The following fields should be included in the call (in addition to the credentials)

KeyNum: [SiteNum]
KeyType: "Site"
ContactNum: -1
Authority: [for contacts: 10; for subscribers: null]

Relation: [Relation value from drop down]
SeqNum: [for contacts: number in order *10 (10, 20, 30, etc); for subscribers: null]
ECVFlag: [if contact is 'Responder' or 'Both' and the 'Dispatch EMS First' is 'No' then set to True, else set to false]
KeysFlag: [HasKeys]
LastName:
FirstName:
ContactType: ['R' for responder; 'N' for notify; 'R/N' for both]

ContactUpdate

Follows the same format as the other update methods. Use the parameters specified for contact insert as well as the update flags for changes. Be sure to send the ContactNum for the contact being updated instead of -1 (as for a new account).

ContactDelete

Removes a contact from the system. Be sure to remove list memberships, phone information, and patient information before deleting the contact

Use the following parameter guidelines for the 'ContactDelete' method

KeyNum=[SiteNum]
KeyType="Site"
ContactNum=[ContactNum]

ContactListMemberList

Return the contact list the contact belongs to. Use the following parameters to return a contact's list membership.

KeyNum=[SiteNum]
ContactNum=[ContactNum]
KeyType='Site'

ContactListMemberInsert

Mark a contact as a member of a list. For responders, each contact must be a member of the 'responders' 'R' contact list. For 'notify' contacts, 'N'. For 'Both', the contact must be part of both 'R' and 'N'.

KeyNum=[SiteNum]
KeyType='Site'
ContactListType=['R' for responder 'N' for notify; add to both lists separately for 'Both']
SeqNum=[integer value beginning with 1 for the order in the list; KeyNum, KeyType, ContactListType and SeqNum are primary key fields in the related table
ContactNum=[ContactNum]

ContactListMemberUpdate

Update a contact list member to change the sequence, etc. See ContactListMemberInsert section for notes

ContactListMemberDelete

Delete a contact from membership in a list. See ContactListMemberInsert for notes and parameter guidelines.

ContactPatientDetail

```
Dim cpd As Stages.ContactPatientDetailQueryResult =
stagesapi.ContactPatientDetail(si.SessionNum, si.SessionPassword, [ContactNum])
```

ContactPatientUpdate

Handles Inserting, Updating and Deleting basic contact medic al information. Use for date of birth:

ColumnDateOfBirth: [Date of Birth]
ChangeFlagDateOfBirth: True

Email Select, Insert, Update and Delete

EmailList

Returns all email addresses associated with a contact or other parent object.

EmailInsert

Inserts an email for a contact or other parent object

KeyNum=[ContactNum]
KeyType='Contact'
SeqNum=[sequence of email for the parent object (must be unique for the parent)]
EmailAddress=[Email Address]
AutoNotifyFlag=[True if want to be notified via email; False otherwise]
DataChangeNotifyFlag=False

EmailUpdate

Updates a given email address. See EmailInsert for parameter usage

EmailDelete

Deletes an email address given the primary key fields KeyNum, KeyType and SeqNum. Can use EmailDeleteList to pass an array of primary fields for deletion.

Name of Person Making Changes

We will log the name of the person who requested the changes via the CSR in the account history. This is achieved through the following call:

```
Dim loa As Stages.LogOperatorActionQueryResult =
stagesapi.LogOperatorAction(si.SessionNum, si.SessionPassword, [AppUserNum], [SiteNum],
[DevNum], TransmitterCode:=[TransmitterCode], OpAct:="XtChangeLog",
OpActComment=[Configurable text that includes the name provided on the form],
NoStatusChangeFlag:=True)
```

Note the configurable text that allows for insertion of the requesting contact's name.

Lookup API Methods

Relation Options:

RelationList

```
Dim rel As Stages.RelationListQueryResult = stagesapi.RelationList(si.SessionNum,
si.SessionPassword)
```

Phone Type Options:

PhoneTypeList

```
Dim ptl As Stages.PhoneTypeListQueryResult = stagesapi.PhoneTypeList(si.SessionNum,
si.SessionPassword)
```

Only 3 phone types are available for the UI:

- Cell (value of 'C')
- Home (value of 'H')
- Work (value of 'W')

Time Zone:

TimeZonePickList

```
Dim tz As Stages.TimeZonePickListQueryResult = stagesapi.TimeZonePickList(si.SessionNum,
si.SessionPassword)
```

Zone (SignalRule)

DeviceConfigList—return a list of all SignalRule
DeviceConfigInsert—insert a SignalRule
DeviceConfigUpdate—update a SignalRule
DeviceConfigDelete—delete a SignalRule

User API Methods

To find out what site group a user is signed in under, which will determine the package used:

```
Dim li As Stages.XtLoginInfoQueryResult = stagesapi.XtLoginInfo(si.SessionNum,
si.SessionPassword)
```

The SiteGroupNum parameter in the reply determines the package (see Preferences section)

To determine permissions for a user check the following:

```
Dim uip As Stages.App2PermissionListQueryResult =
stagesapi.App2PermissionList(si.SessionNum, si.SessionPassword)
```

To determine what can be done in the app look if 'AllowedFlag' is set to true for the following 'Securable'

Permissions if 'AllowedFlag' is true	Securable
Change status of account (In/Out of Service)	XtAppOOS
Edit account information	XtAppWrite
Create/Edit Display Properties	XtAdmin

History

Querying history is relatively straightforward. The challenge is filtering through what is returned. The history call is as follows:

```
Dim hqr As Stages.HistoryQueryResult = stagesapi.History(si.SessionNum, si.SessionPassword,
[sitenum], Nothing, Nothing, Nothing, [InServiceDate], False)
```

Where ‘SiteNum’ and ‘InServiceDate’ come from the ‘DeviceDetail’ method.